Postel

Network Working Group
Request for Comments: 1008

Wayne McCoy
June 1987

IMPLEMENTATION GUIDE

FOR THE

ISO TRANSPORT PROTOCOL

Status of this Memo

This RFC is being distributed to members of the Internet community in order to solicit comments on the Implementors Guide. While this document may not be directly relevant to the research problems of the Internet, it may be of some interest to a number of researchers and implementors. Distribution of this memo is unlimited.

IMPLEMENTATION GUIDE FOR THE ISO TRANSPORT PROTOCOL

1 Interpretation of formal description.

It is assumed that the reader is familiar with both the formal description technique, Estelle [ISO85a], and the transport protocol as described in IS 8073 [ISO84a] and in N3756 [ISO85b].

1.1 General interpretation guide.

The development of the formal description of the ISO Transport Protocol was guided by the three following assumptions.

1. A generality principle

The formal description is intended to express all of the behavior that any implementation is to demonstrate, while not being bound to the way that any particular implementation would realize that behavior within its operating context.

2. Preservation of the deliberate nondeterminism of IS 8073

The text description in the IS 8073 contains deliberate expressions of nondeterminism and indeterminism in the behavior of the transport protocol for the sake of flexibility in application. (Nondeterminism in this context means that the order of execution for a set of actions that can be taken is not specified. Indeterminism means that the execution of a given action cannot be predicted on the basis of system state or the executions of other actions.)

3. Discipline in the usage of Estelle

A given feature of Estelle was to be used only if the nature of the mechanism to be described strongly indicates its usage, or to adhere to the generality principle, or to retain the nondeterminism of IS 8073.

Implementation efficiency was not a particular goal nor was there an attempt to directly correlate Estelle mechanisms and features to implementation mechanisms and features. Thus, the description does not represent optimal behavior for the implemented protocol.

These assumptions imply that the formal description contains higher levels of abstraction than would be expected in a description for a particular operating environment. Such abstraction is essential, because of the diversity of networks and network elements by which implementation and design decisions are influenced. Even when operating environments are essentially identical, design choice and originality in solving a technical problem must be allowed. The same behavior may be expressed in many different ways. The goal in producing the transport formal description was to attempt to capture this equivalence. Some mechanisms of transport are not fully described or appear to be overly complicated because of the adherence to the generality principle. Resolution of these situations may require significant effort on the part of the implementor.

Since the description does not represent optimal behavior for the implemented protocol, implementors should take the three assumptions above into account when using the description to implement the protocol. It may be advisable to adapt the standard description in such a way that:

a. abstractions (such as modules, channels, spontaneous transitions and binding comments) are interpreted and realized as mechanisms appropriate to the operating environment and service requirements;

b. modules, transitions, functions and procedures containing material irrelevant to the classes or options to be supported are reduced or eliminated as needed; and

c. desired real-time behavior is accounted for.

The use in the formal description of an Estelle feature (for instance, "process"), does not imply that an implementation must necessarily realize the feature by a synonymous feature of the operating context. Thus, a module declared to be a "process" in an Estelle description need not represent a real process as seen by a host operating system; "process" in Estelle refers to the

synchronization properties of a set of procedures (transitions).

Realizations of Estelle features and mechanisms are dependent in an essential way upon the performance and service an implementation is to provide. Implementations for operational usage have much more stringent requirements for optimal behavior and robustness than do implementations used for simulated operation (e.g., correctness or conformance testing). It is thus important that an operational implementation realize the abstract features and mechanisms of a formal description in an efficient and effective manner.

For operational usage, two useful criteria for interpretation of formal mechanisms are:

[1] minimization of delays caused by the mechanism itself; e.g.,

--transit delay for a medium that realizes a channel

--access delay or latency for channel medium

--scheduling delay for timed transitions (spontaneous transitions with delay clause)

--execution scheduling for modules using exported variables (delay in accessing variable)

[2] minimization of the "handling" required by each invocation of the mechanism; e.g.,

--module execution scheduling and context switching

--synchronization or protocols for realized channel

--predicate evaluation for spontaneous transitions

Spontaneous transitions represent nondeterminism and indeterminism, so that uniform realization of them in an implementation must be questioned as an implementation strategy. The time at which the action described by a spontaneous transition will actually take place cannot be specified because of one or more of the following situations:

a. it is not known when, relative to any specific event defining the protocol (e.g., input network, input from user, timer

expirations), the conditions enabling the transition will actually occur;

b. even if the enabling conditions are ultimately deterministic, it is not practical to describe all the possible ways this could occur, given the different ways in which implementations will examine these conditions; and

c. a particular implementation may not be concerned with the enabling conditions or will account for them in some other way; i.e., it is irrelevant when the action takes place, if ever.

As an example of a), consider the situation when splitting over the network connection, in Class 4, in which all of the network connections to which the transport connection has been assigned have all disconnected, with the transport connection still in the OPEN state. There is no way to predict when this will happen, nor is there any specific event signalling its occurrence. When it does occur, the transport protocol machine may want to attempt to obtain a new network connection.

As an example of b), consider that timers may be expressed succinctly in Estelle by transitions similar to the following:

```
from A to B
provided predicate delay( timer_interval )

begin
(* action driven by timeout *)
end;
```

But there are operations for which the timer period may need to be very accurate (close to real time) and others in which some delay in executing the action can be tolerated. The implementor must determine the optimal behavior desired for each instance and use an appropriate mechanism to realize it, rather than using a uniform approach to implementing all spontaneous transitions.

As an example of the situation in c), consider the closing of an unused network connection. If the network is such that the cost of letting the network connection remain open is small compared cost of opening it, then an implementation might not want to consider closing the network connection until, say, the weekend. Another implementation might decide to close the network connection within 30 msec after discovering that the connection is not busy. For still another implementation, this could be

meaningless because it operates over a connectionless network service.

If a description has only a very few spontaneous transitions, then it may be relatively easy to implement them literally (i.e., to schedule and execute them as Estelle abstractly does) and not incur the overhead from examining all of the variables that occur in the enabling conditions. However, the number and complexity of the enabling conditions for spontaneous transitions in the transport description strongly suggests that an implementation which realizes spontaneous transitions literally will suffer badly from such overhead.

1.2 Guide to the formal description.

So that implementors gain insight into interpretation of the mechanisms and features of the formal description of transport, the following paragraphs discuss the meanings of such mechanisms and features as intended by the editors of the formal description.

1.2.1 Transport Protocol Entity.

1.2.1.1 Structure.

The diagram below shows the general structure of the Transport Protocol Entity (TPE) module, as given in the formal description. >From an abstract operational viewpoint, the transport protocol Machines (TPMs) and the Slaves operate as child processes of the the TPE process. Each TPM represents the endpoint actions of the protocol on a single transport connection. The Slave represents control of data output to the network. The internal operations of the TPMs and the Slave are discussed below in separate sections.

This structure permits describing multiple connections, multiplexing and splitting on network connections, dynamic existence of endpoints and class negotiation. In the diagram, interaction points are denoted by the symbol "O", while (Estelle) channels joining these interaction points are denoted by

```
          *
          *
          *
```

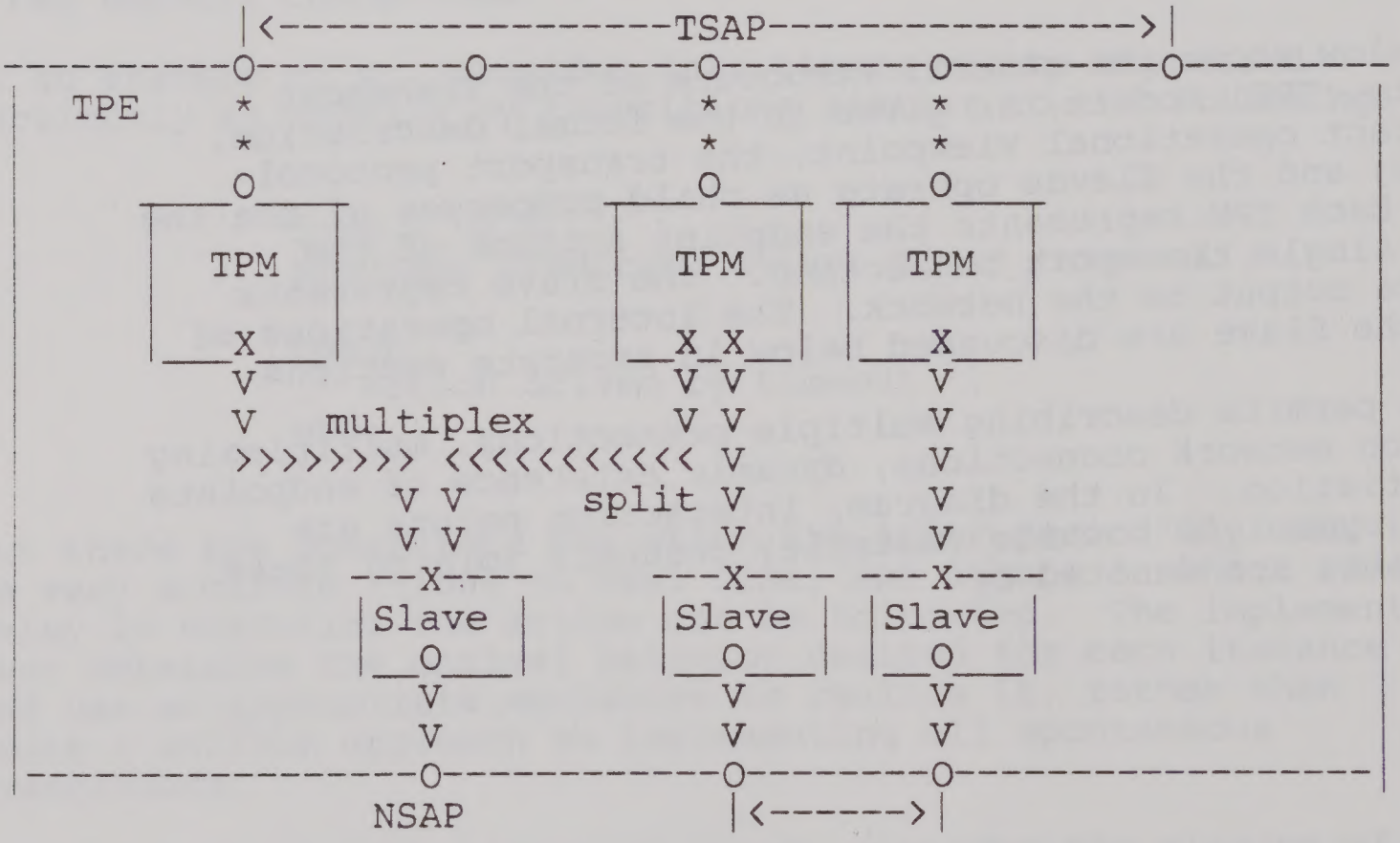
The symbol "X" represents a logical association through variables, and the denotations

```
       <<<<<<<

       >>>>>>>

          V
          V
          V
```

indicate the passage of data, in the direction of the symbol vertices, by way of these associations. The acronyms TSAP and NSAP denote Transport Service Access Point and Network Service Access Point, respectively. The structure of the TSAPs and NSAPs shown is discussed further on, in Parts 1.2.2.1 and 1.2.2.2.



The structuring principles of Estelle provide a formal means of expressing and enforcing certain synchronization properties between communicating processes. It must be stressed that the scheduling implied by Estelle descriptions need not and in some cases should not be implemented. The intent of the structure in the transport formal description is to state formally the synchronization of access tovariables shared by the transport entity and the transport connection endpoints and to permit expression of dynamic objects within the entity. In nearly all aspects of operation except these, it may be more efficient in some implementation environments to permit the TPE and the TPMs to run in parallel (the Estelle scheduling specifically excludes the parallel operation of the TPE and the TPMs). This is particularly true of internal management ("housekeeping") actions and those actions not directly related to communication between the TPE and the TPMs or instantiation of TPMs. Typical actions of this latter sort are: receipt of NSDUs from the network, integrity checking and decoding of TPDUs, and network connection management. Such actions could have been collected into other modules for scheduling closer to that of an implementation, but surely at the risk of further complicating the description. Consequently, the formal description structure should be understood as expressing relationships among actions and objects and not explicit implementation behavior.

1.2.1.2 Transport protocol entity operation.

The details of the operation of the TPE from a conceptual point of view are given in the SYS section of the formal description. However, there are several further comments that can be made regarding the design of the TPE. The Estelle body for the TPE module has no state variable. This means that any transition of the TPE may be enabled and executed at any time. Choice of transition is determined primarily by priority. This suggests that the semantics of the TPE transitions is that of interrupt traps.

The TPE handles only the T-CONNECT-request from the user and the TPM handle all other user input. All network events are handled by the TPE, in addition to resource management to the extent defined in the description. The TPE also manages all aspects of connection references, including reference freezing. The TPE does not explicitly manage the CPU resource for the TPMs, since this is implied by the Estelle scheduling across the module hierarchy. Instantiation of TPMs is also the responsibility of the TPE, as is TPM release when the transport connection is to be closed. Once a TPM is created, the TPE does not in general interfere with TPM's activities, with the following exceptions: the TPE may reduce credit to a Class 4 TPM without notice; the TPE may dissociate a Class 4 TPM from a network connection when splitting is being used. Communication between the TPE and the TPMs is through a set of exported variables owned by the TPMs, and through a channel which

passes TPDUs to be transmitted to the remote peer. This channel is not directly connected to any network connection, so each interaction on it carries a reference number indicating which network connection is to be used. Since the reference is only a reference, this permits usage of this mechanism when the network service is connectionless, as well. The mechanism provides flexibility for both splitting and multiplexing on network connections.

One major function that the TPE performs for all its TPMs is that of initial processing of received TPDUs. First, a set of integrity checks is made to determine if each TPDU in an NSDU is decodable:

a. PDU length indicators and their sums are checked against the NSDU length for consistency;

b. TPDU types versus minimum header lengths for the types are checked, so that if the TPDU can be decoded, then proper association to TPMs can be made without any problem;

c. TPDUs are searched for checksums and the local checksum is computed for any checksum found; and

d. parameter codes in variable part of headers are checked where applicable.

These integrity checks guarantee that an NSDU passing the check can be separated as necessary into TPDUs, these TPDUs can be associated to the transport connections or to the Slave as appropriate and they can be further decoded without error.

The TPE next decodes the fixed part of the TPDU headers to determine the disposition of the TPDU. The Slave gets TPDUs that cannot be assigned to a TPM (spurious TPDU). New TPMs are created in response to CR TPDUs that correspond to a TSAP for this TPE.

All management of NSAPs is done by the TPE. This consists of keeping track of all network connections, their service quality characteristics and their availability, informing the TPMs associated with these network connections.

The TPE has no timer module as such. Timing is handled by using the DELAY feature of Estelle, since this feature captures the essence of timing without specifying how the actual timing is to be achieved within the operating environment. See Part 1.2.5 for more details.

1.2.2 Service Access Points.

The service access points (SAP) of the transport entity are modeled using the Estelle channel/interaction point formalism. (Note: The

term "channel" in Estelle is a keyword that denotes a set of interactions which may be exchanged at interaction points [LIN85]. However, it is useful conceptually to think of "channel" as denoting a communication path that carries the interactions between modules.) The abstract service primitives for a SAP are interactions on channels entering and leaving the TPE. The transport user is considered to be at the end of the channel connected to the transport SAP (TSAP) and the network service provider is considered to be at the end of the channel connected to the network SAP (NSAP). An interaction put into a channel by some module can be considered to move instantaneously over the channel onto a queue at the other end. The sender of such an interaction no longer has access to the interaction once it has been put into the channel. The operation of the system modeled by the formal description has been designed with this semantics in mind, rather than the equivalent but much more abstract Estelle semantics. (In the Estelle semantics, each interaction point is considered to have associated with it an unbounded queue. The "attach" and "connect" primitives bind two interaction points, such that an action, implied by the keyword "out", at one interaction point causes a specified interaction to be placed onto the queue associated with the other interaction point.) The sections that follow discuss the TSAP and the NSAP and the way that these SAPs are described in the formal description.

1.2.2.1 Transport Service Access Point.

The international transport standard allows for more than one TSAP to be associated with a transport entity, and multiple users may be associated with a given TSAP. A situation in which this is useful is when it is desirable to have a certain quality of service correlated with a given TSAP. For example, one TSAP could be reserved for applications requiring a high throughput, such as file transfer. The operation of transport connections associated with this TSAP could then be designed to favor throughput. Another TSAP might serve users requiring short response time, such as terminals. Still another TSAP could be reserved for encryption reasons.

In order to provide a way of referencing users associated with TSAPs, the user access to transport in the formal description is through an array of Estelle interaction points. This array is indexed by a TSAP address (T_address) and a Transport Connection Endpoint Identifier (TCEP_id). Note that this dimensional object (TSAP) is considered simply to be a uniform set of abstract interfaces. The indices must be of (Pascal) ordinal type in Estelle. However, the actual address structure of TSAPs may not conform easily to such typing in an implementation. Consequently, the indices as they appear in the formal description should be viewed as an organizational mechanism rather than as an explicit way of associating objects in an operational setting. For example, actual TSAP addresses might be kept in some kind of table, with the table index being used to reference objects associated with the TSAP.

One particular issue concerned with realizing TSAPs is that of making known to the users the means of referencing the transport interface, i.e., somehow providing the T_addresses and TCEP_ids to the users. This issue is not considered in any detail by either IS 7498 [ISO84b] or IS 8073. Abstractly, the required reference is the T_address/TCEP_id pair. However, this gives no insight as to how the mechanism could work. Some approaches to this problem are discussed in Part 5.

Another issue is that of flow control on the TSAP channels. Flow control is not part of the semantics for the Estelle channel, so the problem must be dealt with in another way. The formal description gives an abstract definition of interface flow control using Pascal and Estelle mechanisms. This abstraction resembles many actual schemes for flow control, but the realization of flow control will still be dependent on the way the interface is implemented. Part 3.2 discusses this in more detail.

1.2.2.2 Network Service Access Point.

An NSAP may also have more than one network connection associated with it. For example, the virtual circuits of X.25 correspond with this notion. On the other hand, an NSAP may have no network connection associated with it, for example when the service at the NSAP is connectionless. This certainly will be the case when transport operates on a LAN or over IP. Consequently, although the syntactical appearance of the NSAP in the formal description is similar to that for the TSAP, the semantics are essentially distinct [NTI85].

Distinct NSAPs can correspond or not to physically distinct networks. Thus, one NSAP could access X.25 service, another might access an IEEE 802.3 LAN, while a third might access a satellite link. On the other hand, distinct NSAPs could correspond to different addresses on the same network, with no particular rationale other than facile management for the distinction. There are performance and system design issues that arise in considering how NSAPs should be managed in such situations. For example, if distinct NSAPs represent distinct networks, then a transport entity which must handle all resource management for the transport connections and operate these connections as well may have trouble keeping pace with data arriving concurrently from two LANs and a satellite link. It might be a better design solution to separate the management of the transport connection resources from that of the NSAP resources and inputs, or even to provide separate transport entities to handle some of the different network services, depending on the service quality to be maintained. It may be helpful to think of the (total) transport service as not necessarily being provided by a single monolithic entity--several distinct entities can reside at the transport layer on the same end-system.

The issues of NSAP management come primarily from connection-oriented network services. This is because a connectionless service is either available to all transport connections or it is available to none, representing infinite degrees of multiplexing and splitting. In the connection-oriented case, NSAP management is complicated by multiplexing, splitting, service quality considerations and the particular character of the network service. These issues are discussed further in Part 3.4.1. In the formal description, network connection management is carried out by means of a record associated with each possible connection and an array, associated with each TPM, each array member corresponding to a possible network connection. Since there is, on some network services, a very large number of possible network connections, it is clear that in an implementation these data structures may need to be made dynamic rather than static. The connection record, indexed by NSAP and NCEP_id, consists of a Slave module reference, virtual data connections to the TPMs to be associated with the network connection, a data connection (out) to the NSAP, and a data connection to the Slave. There is also a "state" variable for keeping track of the availability of the connection, variables for managing the Slave and an internal reference number to identify the connection to TPMs. A member of the network connection array associated with a TPM provides the TPM with status information on the network connection and input data (network) events and TPDUs). A considerable amount of management of the network connections is provided by the formal description, including splitting, multiplexing, service quality (when defined), interface flow control, and concatenation of TPDUs. This management is carried out solely by the transport entity, leaving the TPMs free to handle only the explicit transport connection issues. This management scheme is flexible enough that it can be simplified and adapted to handle the NSAP for a connectionless service.

The principal issue for management of connectionless NSAPs is that of buffering, particularly if the data transmission rates are high, or there is a large number of transport connections being served. It may also be desirable for the transport entity to monitor the service it is getting from the network. This would entail, for example, periodically computing the mean transmission delays for adjusting timers or to exert backpressure on the transport connections if network access delay rises, indicating loading. (In the formal description, the Slave processor provides a simple form of output buffer management: when its queue exceeds a threshold, it shuts off data from the TPMs associated with it. Through primitive functions, the threshold is loosely correlated with network behavior. However, this mechanism is not intended to be a solution to this difficult performance problem.)

1.2.3 Transport Protocol Machine.

Transport Protocol Machines (TPM) in the formal description are in six classes: General, Class 0, Class 1, Class 2, Class 3 and Class 4. Only the General, Class 2 and Class 4 TPMs are discussed here. The reason for this diversity is to facilitate describing class negotiations and to show clearly the actions of each class in the data transfer phase. The General TPM is instantiated when a connection request is received from a transport user or when a CR TPDU is received from a remote peer entity. This TPM is replaced by a class-specific TPM when the connect response is received from the responding user or when the CC TPDU is received from the responding peer entity.

The General, Class 2 and Class 4 TPMs are discussed below in more detail. In an implementation, it probably will be prudent to merge the Class 2 and Class 4 operations with that of the General TPM, with new variables selecting the class-specific operation as necessary (see also Part 9.4 for information on obtaining Class 2 operation from a Class 4 implementation). This may simplify and improve the behavior of the implemented protocol overall.

1.2.3.1 General Transport Protocol Machine.

Connection negotiation and establishment for all classes can be handled by the General Transport Protocol Machine. Some parts of the description of this TPM are sufficiently class dependent that they can safely be removed if that class is not implemented. Other parts are general and must be retained for proper operation of the TPM. The General TPM handles only connection establishment and negotiation, so that only CR, CC, DR and DC TPDUs are sent or received (the TPE prevents other kinds of TPDUs from reaching the General TPM).

Since the General TPM is not instantiated until a T-CONNECT-request or a CR TPDU is received, the TPE creates a special internal connection to the module's TSAP interaction point to pass the T-CONNECT-request event to the TPM. This provides automaton completeness according to the specfication of the protocol. When the TPM is to be replaced by a class-specific TPM, the sent or received CC is copied to the new TPM so that negotiation information is not lost.

In the IS 8073 state tables for the various classes, the majority of the behavioral information for the automaton is contained in the connection establishment phase. The editors of the formal description have retained most of the information contained in the state tables of IS 8073 in the description of the General TPM.

1.2.3.2 Class 2 Transport Protocol Machine.

The formal description of the Class 2 TPM closely resembles that of

Class 4, in many respects. This is not accidental, in that: the conformance statement in IS 8073 links Class 2 with Class 4; and the editors of the formal description produced the Class 2 TPM description by copying the Class 4 TPM description and removing material on timers, checksums, and the like that is not part of the Class 2 operation. The suggestion of obtaining Class 2 operation from a Class 4 implementation, described in Part 9.4, is in fact based on this adaptation.

One feature of Class 2 that does not appear in Class 4, however, is the option to not use end-to-end flow control. In this mode of operation, Class 2 is essentially Class 0 with multiplexing. In fact, the formal description of the Class 0 TPM was derived from Class 2 (in IS 8073, these two classes have essentially identical state tables). This implies that Class 0 operation could be obtained from Class 2 by not multiplexing, not sending DC TPDUs, electing not to use flow control and terminating the network connection when a DR TPDU is received (expedited data cannot be used if flow control is not used). When Class 2 is operated in this mode, a somewhat different procedure is used to handle data flow internal to the TPM than is used when end-to-end flow control is present.

1.2.3.3 Class 4 Transport Protocol Machine.

Dynamic queues model the buffering of TPDUs in both the Class 4 and Class 2 TPMs. This provides a more general model of implementations than does the fixed array representation and is easier to describe. Also, the fixed array representation has semantics that, carried into an implementation, would produce inefficiency. Consequently, linked lists with queue management functions make up the TPDU storage description, despite the fact that pointers have a very implementation-like flavor. One of the queue management functions permits removing several TPDUs from the head of the send queue, to model the acknowledgement of several TPDUs at once, as specified in IS 8073. Each TPDU record in the queue carries the number of retransmissions tried, for timer control (not present in the Class 2 TPDU records).

There are two states of the Class 4 TPM that do not appear in IS 8073. One of these was put in solely to facilitate obtaining credit in case no credit was granted for the CR or CC TPDU. The other state was put in to clarify operations when there is unacknowledged expedited data outstanding (Class 2 does not have this state).

The timers used in the Class 4 TPM are discussed below, as is the description of end-to-end flow control.

For simplicity in description, the editors of the formal description assumed that no queueing of expedited data would occur at the user interface of the receiving entity. The user has the capability to block the up-flow of expedited data until it is ready. This

assumption has several implications. First, an ED TPDU cannot be acknowledged until the user is ready to accept it. This is because the receipt of an EA TPDU would indicate to the sending peer that the receiver is ready to receive the next ED TPDU, which would not be true. Second, because of the way normal data flow is blocked by the sending of an ED TPDU, normal data flow ceases until the receiving user is ready for the ED TPDU. This suggests that the user interface should employ separate and noninterfering mechanisms for passing normal and expedited data to the user. Moreover, the mechanism for expedited data passage should be blocked only in dire operational conditions. This means that receipt of expedited data by the user should be a procedure (transition) that operates at nearly the highest priority in the user process. The alternative to describing the expedited data handling in this way would entail a scheme of properly synchronizing the queued ED TPDUs with the DT TPDUs received. This requires some intricate handling of DT and ED sequence numbers. While this alternative may be attractive for implementations, for clarity in the formal description it provides only unnecessary complication.

The description of normal data TSDU processing is based on the assumption that the data the T-DATA-request refers to is potentially arbitrarily long. The semantic of the TSDU in this case is analogous to that of a file pointer, in the sense that any file pointer is a reference to a finite but arbitrarily large set of octet-strings. The formation of TPDUs from this string is analogous to reading the file in fixed-length segments--records or blocks, for example. The reassembly of TPDUs into a string is analogous to appending each TPDU to the tail of a file; the file is passed when the end-of-TSDU (end-of-file) is received. This scheme permits conceptual buffering of the entire TSDU in the receiver and avoids the question of whether or not received data can be passed to the user before the EOT is received. (The file pointer may refer to a file owned by the user, so that the question then becomes moot.)

The encoding of TPDUs is completely described, using Pascal functions and some special data manipulation functions of Estelle (these are not normally part of Pascal). There is one encoding function corresponding to each TPDU type, rather than a single parameterized function that does all of them. This was done so that the separate structures of the individual types could be readily discerned, since the purpose of the functions is descriptive and not necessarily computational.

The output of TPDUs from the TPM is guarded by an internal flow control flag. When the TPDU is first sent, this flag is ignored, since if the TPDU does not get through, a retransmission may take care of it. However, when a retransmission is tried, the flag is heeded and the TPDU is not sent, but the retransmission count is incremented. This guarantees that either the TPDU will eventually be sent or the connection will time out (this despite the fact that

the peer will never have received any TPDU to acknowledge). Checksum computations are done in the TPM rather than by the TPE, since the TPE must handle all classes. Also, if the TPMs can be made to truly run in parallel, the performance may be greatly enhanced.

The decoding of received TPDUs is partially described in the Class 4 TPM description. Only the CR and CC TPDUs present any problems in decoding, and these are largely due to the nondeterministic order of parameters in the variable part of the TPDU headers and the locality-and class-dependent content of this variable part. Since contents of this variable part (except the TSAP-IDs) do not affect the association of the TPDU with a transport connection, the decoding of the variable part is not described in detail. Such a description would be very lengthy indeed because of all the possibilities and would not contribute measurably to understanding by the reader.

1.2.4 Network Slave.

The primary functions of the Network Slave are to provide downward flow control in the TPE, to concatenate TPDUs into a single NSDU and to respond to the receipt of spurious TPDUs. The Slave has an internal queue on which it keeps TPDUs until the network is ready to accept them for transmission. The TPE is kept informed as to the length of queue, and the output of the TPMs is throttled if the length exceeds this some threshold. This threshold can be adjusted to meet current operating conditions. The Slave will concatenate the TPDUs in its queue if the option to concatenate is exercised and the conditions for concatenating are met. Concatenation is a TPE option, which may be exercised or not at any time.

1.2.5 Timers.

In the formal description timers are all modeled using a spontaneous transition with delay, where the delay parameter is the timer period. To activate the timer, a timer identifier is placed into a set, thereby satisfying a predicate of the form

provided timer_x in active_timers

However, the transition code is not executed until the elapsed time ;from the placement of the identifier in the set is at least equal to the delay parameter. The editors of the formal description chose to model timers in this fashion because it provided a simply expressed description of timer behavior and eliminated having to consider how timing is done in a real system or to provide special timer modules and communication to them. It is thus recommended that implementors not follow the timer model closely in implementations, considering instead the simplest and most efficient means of timing permitted by the implementation environment. Implementors should

also note that the delay parameter is typed "integer" in the formal description. No scale conversion from actual time is expressed in the timer transition, so that this scale conversion must be considered when timers are realized.

1.2.5.1 Transport Protocol Entity timers.

There is only one timer given in the formal description of the TPE--the reference timer. The reference timer was placed here ;so that it can be used by all classes and all connections, as needed. There is actually little justification for having a reference timer within the TPM--it wastes resources by holding the transport endpoint, even though the TPM is incapable of responding to any input. Consequently, the TPE is responsible for all aspects of reference management, including the timeouts.

1.2.5.2 Transport Protocol Machine timers.

Class 2 transport does not have any timers that are required by IS 8073. However, the standard does recommend that an optional timer be used by Class 2 in certain cases to avoid deadlock. The formal description provides this timer, with comments to justify its usage. It is recommended that such a timer be provided for Class 2 operation. Class 4 transport has several timers for connection control, flow control and retransmissions of unacknowledged data. Each of these timers is discussed briefly below in terms of how they were related to the Class 4 operations in the formal description. Further discussion of these timers is given in Part 8.

1.2.5.2.1 Window timer.

The window timer is used for transport connection control as well as providing timely updates of flow control credit information. One of these timers is provided in each TPM. It is reset each time an AK TPDU is sent, except during fast retransmission of AKs for flow control confirmation, when it is disabled.

1.2.5.2.2 Inactivity timer.

The primary usage of the inactivity timer is to detect when the remote peer has ceased to send anything (including AK TPDUs). This timer is mandatory when operating over a connectionless network service, since there is no other way to determine whether or not the remote peer is still functioning. On a connection-oriented network service it has an additional usage since to some extent the continued existence of the network connection indicates that the peer host has not crashed.

Because of splitting, it is useful to provide an inactivity timer on each network connection to which a TPM is assigned. In this manner, if a network connection is unused for some time, it can be released,

even though a TPM assigned to it continues to operate over other network connections. The formal description provides this capability in each TPM.

1.2.5.2.3 Network connection timer.

This timer is an optional timer used to ensure that every network connection to which a TPM is assigned gets used periodically. This prevents the expiration of the peer entity's inactivity timer for a network connection. There is one timer for each network connection to which the TPM is assigned. If there is a DT or ED TPDU waiting to be sent, then it is chosen to be sent on the network connection. If no such TPDU is waiting, then an AK TPDU is sent. Thus, the NC timer serves somewhat the same purpose as the window timer, but is broader in scope.

1.2.5.2.4 Give-up timer.

There is one give-up timer for a TPM which is set whenever the retransmission limit for any CR, CC, DT, ED or DR TPDU is reached. Upon expiration of this timer, the transport connection is closed.

1.2.5.2.5 Retransmission timers.

Retransmission timers are provided for CR, CC, DT, ED and DR TPDUs. The formal description provides distinct timers for each of these TPDU types, for each TPM. However, this is for clarity in the description, and Part 8.2.5 presents arguments for other strategies to be used in implementations. Also, DT TPDUs with distinct sequence numbers are each provided with timers, as well. There is a primitive function which determines the range within the send window for which timers will be set. This has been done to express flexibility in the retransmission scheme.

The flow control confirmation scheme specified in IS 8073 also provides for a "fast" retransmission timer to ensure the reception of an AK TPDU carrying window resynchronization after credit reduction or when opening a window that was previously closed. The formal description permits one such timer for a TPM. It is disabled after the peer entity has confirmed the window information.

1.2.5.2.6 Error transport protocol data unit timer.

In IS 8073, there is a provision for an optional timeout to limit the wait for a response by the peer entity to an ER TPDU. When this timer expires, the transport connection is terminated. Each Class 2 or Class 4 TPM is provided with one of these timers in N3756.

1.2.6 End-to-end Flow Control.

Flow control in the formal description has been written in such a way

as to permit flexibility in credit control schemes and acknowledgement strategies.

1.2.6.1 Credit control.

The credit mechanism in the formal description provides for actual management of credit by the TPE. This is done through variables exported by the TPMs which indicate to the TPE when credit is needed and for the TPE to indicate when credit has been granted. In this manner, the TPE has control over the credit a TPM has. The mechanism allows for reduction in credit (Class 4 only) and the possibility of precipitous window closure. The mechanism does not preclude the use of credit granted by the user or other sources, since credit need is expressed as current credit being less than some threshold. Setting the threshold to zero permits these other schemes. An AK TPDU is sent each time credit is updated.

The end-to-end flow control is also coupled to the interface flow control to the user. If the user has blocked the interface up-flow, then the TPM is prohibited from requesting more credit when the current window is used up.

1.2.6.2 Acknowledgement.

The mechanism for acknowledging normal data provides flexibility sufficient to send an AK TPDU in response to every Nth DT TPDU received where N > 0 and N may be constant or dynamically determined. Each TPM is provided with this, independent of all other TPMs, so that acknowledgement strategy can be determined separately for each transport connection. The capability of altering the acknowledgement strategy is useful in operation over networks with varying error rates.

1.2.6.3 Sequencing of received data.

It is not specified in IS 8073 what must be done with out-of-sequence but within-window DT TPDUs received, except that an AK TPDU with current window and sequence information be sent. There are performance reasons why such DT TPDUs should be held (cached): in particular, avoidance of retransmissions. However, this buffering scheme is complicated to implement and worse to describe formally without resorting to mechanisms too closely resembling implementation. Thus, the formal description mechanism discards such DT TPDUs and relies on retransmission to fill the gaps in the window sequence, for the sake of simplicity in the description.

1.2.7 Expedited data.

The transmission of expedited data, as expressed by IS 8073, requires the blockage of normal data transmission until the acknowledgement is received. This is handled in the formal description by providing a

special state in which normal data transmission cannot take place. However, recent experiments with Class 4 transport over network services with high bandwidth, high transit delay and high error rates, undertaken by the NBS and COMSAT Laboratories, have shown that the protocol suffers a marked decline in its performance in such conditions. This situation has been presented to ISO, with the result that the the protocol will be modified to permit the sending of normal data already accepted by the transport entity from the user before the expedited data request but not yet put onto the network. When the modification is incorporated into IS 8073, the formal description will be appropriately aligned.

2 Environment of implementation.

The following sections describe some general approaches to implementing the transport protocol and the advantages and disadvantages of each. Certain commercial products are identified throughout the rest of this document. In no case does such identification imply the recommendation or endorsement of these products by the Department of Defense, nor does it imply that the products identified are the best available for the purpose described. In all cases such identification is intended only to illustrate the possibility of implementation of an idea or approach. UNIX is a trademark of AT&T Bell Laboratories.

Most of the discussions in the remainder of the document deal with Class 4 exclusively, since there are far more implementation issues with Class 4 than for Class 2. Also, since Class 2 is logically a special case of Class 4, it is possible to implement Class 4 alone, with special provisions to behave as Class 2 when necessary.

2.1 Host operating system program.

A common method of implementing the OSI transport service is to integrate the required code into the specific operating system supporting the data communications applications. The particular technique for integration usually depends upon the structure and facilities of the operating system to be used. For example, the transport software might be implemented in the operating system kernel, accessible through a standard set of system calls. This scheme is typically used when implementing transport for the UNIX operating system. Class 4 transport has been implemented using this technique for System V by AT&T and for BSD 4.2 by several organizations. As another example, the transport service might be structured as a device driver. This approach is used by DEC for the VAX/VMS implementation of classes 0, 2, and 4 of the OSI transport protocol. The Intel iRMX-86 implementation of Class 4 transport is another example. Intel implements the transport software as a first level job within the operating system. Such an approach allows the software to be linked to the operating system and loaded with every

boot of the system.

Several advantages may accrue to the communications user when transport is implemented as an integral part of the operating system. First, the interface to data communications services is well known to the application programmer since the same principles are followed as for other operating system services. This allows the fast implementation of communications applications without the need for retraining of programmers. Second, the operating system can support several different suites of protocols without the need to change application programs. This advantage can be realized only with careful engineering and control of the user-system call interface to the transport services. Third, the transport software may take advantage of the normally available operating system services such as scheduling, flow control, memory management, and interprocess communication. This saves time in the development and maintenance of the transport software.

The disadvantages that exist with operating system integration of the TP are primarily dependent upon the specific operating system. However, the major disadvantage, degradation of host application performance, is always present. Since the communications software requires the attention of the processor to handle interrupts and process protocol events, some degradation will occur in the performance of host applications. The degree of degradation is largely a feature of the hardware architecture and processing resources required by the protocol. Other disadvantages that may appear relate to limited performance on the part of the communications service. This limited performance is usually a function of the particular operating system and is most directly related to the method of interprocess communication provided with the operating system. In general, the more times a message must be copied from one area of memory to another, the poorer the communications software will perform. The method of copying and the number of copies is often a function of the specific operating system. For example, copying could be optimized if true shared memory is supported in the operating system. In this case, a significant amount of copying can be reduced to pointer-passing.

2.2 User program.

The OSI transport service can be implemented as a user job within any operating system provided a means of multi-task communications is available or can be implemented. This approach is almost always a bad one. Performance problems will usually exist because the communication task is competing for resources like any other application program. The only justification for this approach is the need to develop a simple implementation of the transport service quickly. The NBS implemented the transport protocol using this approach as the basis for a transport protocol correctness testing system. Since performance was not a goal of the NBS implementation,

the ease of development and maintenance made this approach attractive.

2.3 Independent processing element attached to a system bus.

Implementation of the transport service on an independent processor that attaches to the system bus may provide substantial performance improvements over other approaches. As computing power and memory have become cheaper this approach has become realistic. Examples include the Intel implementation of iNA-961 on a variety of multibus boards such as the iSBC 186/51 and the iSXM 554. Similar products have been developed by Motorola and by several independent vendors of IBM PC add-ons. This approach requires that the transport software operate on an independent hardware set running under operating system code developed to support the communications software environment. Communication with the application programs takes place across the system bus using some simple, proprietary vendor protocol. Careful engineering can provide the application programmer with a standard interface to the communications processor that is similar to the interface to the input/output subsystem.

The advantages of this approach are mainly concentrated upon enhanced performance both for the host applications and the communications service. Depending on such factors as the speed of the communications processor and the system bus, data communications throughput may improve by one or two orders of magnitude over that available from host operating system integrated implementations. Throughput for host applications should also improve since the communications processing and interrupt handling for timers and data links have been removed from the host processor. The communications mechanism used between the host and communication processors is usually sufficiently simple that no real burden is added to either processor.

The disadvantages for this approach are caused by complexity in developing the communications software. Software development for the communications board cannot be supported with the standard operating system tools. A method of downloading the processor board and debugging the communications software may be required; a trade-off could be to put the code into firmware or microcode. The communications software must include at least a hardware monitor and, more typically, a small operating system to support such functions as interprocess communication, buffer management, flow control, and task synchronization. Debugging of the user to communication subsystem interface may involve several levels of system software and hardware.

The design of the processing element can follow conventional lines, in which a single processor handling almost all of the operation of the protocol. However, with inexpensive processor and memory chips now available, a multiprocessor design is economically viable. The diagram below shows one such design, which almost directly

corresponds to the structure of the formal description. There are several advantages to this design:

1) management of CPU and memory resources is at a minimum;

2) essentially no resource contention;

3) transport connection operation can be written in microcode, separate from network service handling;

4) transport connections can run with true parallelism;

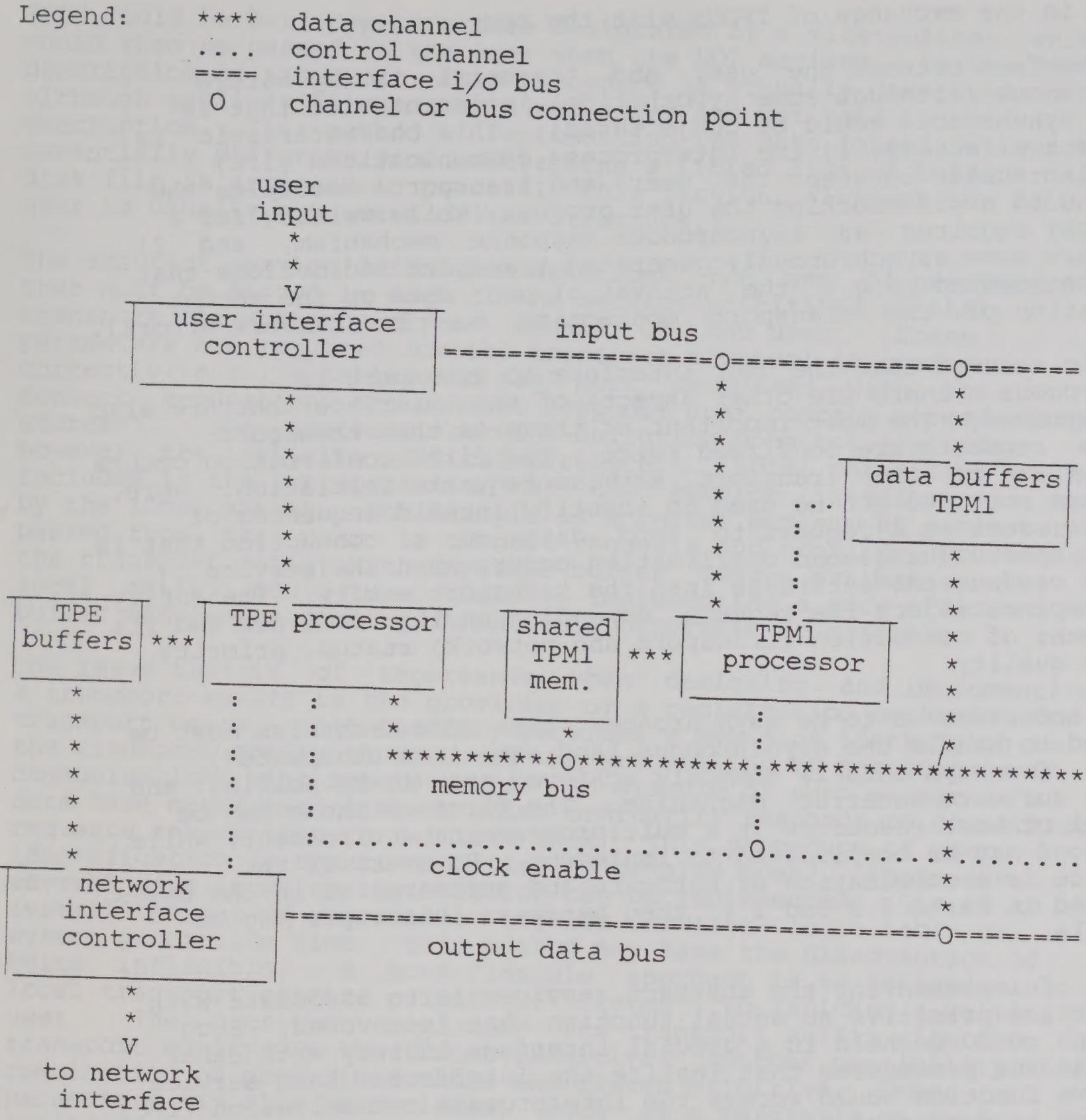
5) throughput is not limited by contention of connections for CPU and network access; and

6) lower software complexity, due to functional separation.

Possible disadvantages are greater inflexibility and hardware complexity. However, these might be offset by lower development costs for microcode, since the code separation should provide overall lower code complexity in the TPE and the TPM implementations.

In this system, the TPE instantiates a TPM by enabling its clock. Incoming Outgoing are passed to the TPMs along the memory bus. TPDUs TPDUs from a TPM are sent on the output data bus. The user interface controller accepts connect requests from the user and directs them to the TPE. The TPE assigns a connection reference and informs the interface controller to direct further inputs for this connection to the designated TPM. The shared TPM memory is analogous to the exported variables of the TPM modules in the formal description, and is used by the TPE to input TPDUs and other information to the TPM.

In summary, the off-loading of communications protocols onto independent processing systems attached to a host processor across a system bus is quite common. As processing power and memory become cheaper, the amount of software off-loaded grows. it is now typical to fine transport service available for several system buses with interfaces to operating systems such as UNIX, XENIX, iRMX, MS-DOS, and VERSADOS.



2.4 Front end processor.

A more traditional approach to off-loading communications protocols involves the use of a free-standing front end processor, an approach very similar to that of placing the transport service onto a board attached to the system bus. The difference is one of scale. Typical front end p interface locally as desirable, as long as such additions are strictly local (i.e., the invoking of such services does not

result in the exchange of TPDUs with the peer entity).

The interface between the user and transport is by nature asynchronous (although some hypothetical implementation that is wholly synchronous could be conjectured). This characteristic is due to two factors: 1) the interprocess communications (IPC) mechanism--used between the user and transport--decouples the two, and to avoid blocking the user process (while waiting for a response) requires an asynchronous response mechanism, and 2) there are some asynchronously-generated transport indications that must be handled (e.g., the arrival of user data or the abrupt termination of the transport connection due to network errors).

If it is assumed that the user interface to transport is asynchronous, there are other aspects of the interface that are also predetermined. The most important of these is that transport service requests are confirmed twice. The first confirmation occurs at the time of the transport service request initiation. Here, interface routines can be used to identify invalid sequences of requests, such as a request to send data on a connection that is not yet open. The second confirmation occurs when the service request crosses the interface into the transport entity. The entity may accept or reject the request, depending on its resources and its assessment of connection (transport and network) status, priority, service quality.

If the interface is to be asynchronous, then some mechanism must be provided to handle the asynchronous (and sometimes unexpected) events. Two ways this is commonly achieved are: 1) by polling, and 2) by a software interrupt mechanism. The first of these can be wasteful of host resources in a multiprogramming environment, while the second may be complicated to implement. However, if the interface is a combination of hardware and software, as in the cases discussed in Parts 2.3 and 2.4, then hardware interrupts may be available.

One way of implementing the abstract services is to associate with each service primitive an actual function that is invoked. Such functions could be held in a special interface library with other functions and procedures that realize the interface. Each service primitive function would access the interprocess communication (IPC) mechanism as necessary to pass parameters to/from the transport entity.

The description of the abstract service in IS 8073 and N3756 implies that the interface must handle TSDUs of arbitrary length. This situation suggests that it may be useful to implement a TSDU as an object such as a file-pointer rather than as the message itself. In this way, in the sending entity, TPDUs can be formed by reading segments of TPDU-size from the file designated, without regard for the actual length of the file. In the receiving entity, each new

TPDU could be buffered in a file designated by a file-pointer, which would then be passed to the user when the EOT arrives. In the formal description of transport, this procedure is actually described, although explicit file-pointers and files are not used in the description. This method of implementing the data interface is not essentially different from maintaining a linked list of buffers. (A disk file is arranged in precisely this fashion, although the file user is usually not aware of the structure.)

The abstract service definition describes the set of parameters that must be passed in each of the service primitives so that transport can act properly on behalf of the user. These parameters are required for the transport protocol to operate correctly (e.g., a called address must be passed with the connect request and the connect response must contain a responding address). The abstract service defintion does not preclude, however, the inclusion of local parameters. Local parameters may be included in the implementation of the service interface for use by the local entity. One example is a buffer management parameter passed from the user in connect requests and confirms, providing the transport entity with expected buffer usage estimates. The local entity could use this in implementing a more efficient buffer management strategy than would otherwise be possible.

One issue that is of importance when designing and implementing a transport entity is the provision of a registration mechanism for transport users. This facility provides a means of identifying to the transport entity those users who are willing to participate in communications with remote users. An example of such a user is a data base management system, which ordinarily responds to connections requests rather than to initiate them. This procedure of user identification is sometimes called a "passive open". There are several ways in which registration can be implemented. One is to install the set of users that provide services in a table at system generation time. This method may have the disadvantage of being inflexible. A more flexible approach is to implement a local transport service primitive, "listen", to indicate a waiting user. The user then registers its transport suffix with the transport entity via the listen primitive. Another possibility is a combination of predefined table and listen primitive. Other parameters may also be included, such as a partially or fully qualified transport address from which the user is willing to receive connections. A variant on this approach is to provide an ACTIVE/PASSIVE local parameter on the connect request service primitive. Part 5 discusses this issue in more detail.

3.2 Flow control.

Interface flow control is generally considered to be a local implementation issue. However, in order to completely specify the behavior of the transport entity, it was necessary to include in the

formal description a model of the control of data flow across the service boundaries of transport. The international standards for transport and the OSI reference model state only that interface flow control shall be provided but give no guidance on its features.

The actual mechanisms used to accomplish flow control, which need not explicitly follow the model in the formal description, are dependent on the way in which the interface itself is realized, i.e., what TSDUs and service primitives really are and how the transport entity actually communicates with its user, its environment, and the network service. For example, if the transport entity communicates with its user by means of named (UNIX) pipes, then flow control can be realized using a special interface library routine, which the receiving process invokes, to control the pipe. This approach also entails some consideration for the capacity of the pipe and blocking of the sending process when the pipe is full (discussed further in Part 3.3). The close correspondence of this interpretation to the model is clear. However, such an interpretation is apparently not workable if the user process and the transport entity are in physically separate processors. In this situation, an explicit protocol between the receiving process and the sending process must be provided, which could have the complexity of the data transfer portion of the Class 0 transport protocol (Class 2 if flow controlled). Note that the formal model, under proper interpretation, also describes this mechanism.

3.3 Interprocess communication.

One of the most important elements of a data communication system is the approach to interprocess communication (IPC). This is true because suites of protocols are often implemented as groups of cooperating tasks. Even if the protocol suites are not implemented as task groups, the communication system is a funnel for service requests from multiple user processes. The services are normally communicated through some interprocess pathway. Usually, the implementation environment places some restrictions upon the interprocess communications method that can be used. This section describes the desired traits of IPC for use in data communications protocol implementations, outlines some possible uses for IPC, and discusses three common and generic approaches to IPC.

To support the implementation of data communications protocols, IPC should possess several desirable traits. First, IPC should be transaction based. This permits sending a message without the overhead of establishing and maintaining a connection. The transactions should be confirmed so that a sender can detect and respond to non-delivery. Second, IPC should support both the synchronous and the asynchronous modes of message exchange. An IPC receiver should be able to ask for delivery of any pending messages and not be blocked from continuing if no messages are present. Optionally, the receiver should be permitted to wait if no messages

are present, or to continue if the path to the destination is congested. Third, IPC should preserve the order of messages sent to the same destination. This allows the use of the IPC without modification to support protocols that preserve user data sequence. Fourth, IPC should provide a flow control mechanism to allow pacing of the sender's transmission speed to that of the receiver.

The uses of IPC in implementation of data communication systems are many and varied. A common and expected use for IPC is that of passing user messages among the protocol tasks that are cooperating to perform the data communication functions. The user messages may contain the actual data or, more efficiently, references to the location of the user data. Another common use for the IPC is implementation and enforcement of local interface flow control. By limiting the number of IPC messages queued on a particular address, senders can be slowed to a rate appropriate for the IPC consumer. A third typical use for IPC is the synchronization of processes. Two cooperating tasks can coordinate their activities or access to shared resources by passing IPC messages at particular events in their processing.

More creative uses of IPC include buffer, timer, and scheduling management. By establishing buffers as a list of messages available at a known address at system initialization time, the potential exists to manage buffers simply and efficiently. A process requiring a buffer would simply read an IPC message from the known address. If no messages (i.e., buffers) are available, the process could block (or continue, as an option). A process that owned a buffer and wished to release it would simply write a message to the known address, thus unblocking any processes waiting for a buffer.

To manage timers, messages can be sent to a known address that represents the timer module. The timer module can then maintain the list of timer messages with respect to a hardware clock. Upon expiration of a timer, the associated message can be returned to the originator via IPC. This provides a convenient method to process the set of countdown timers required by the transport protocol.

Scheduling management can be achieved by using separate IPC addresses for message classes. A receiving process can enforce a scheduling discipline by the order in which the message queues are read. For example, a transport process might possess three queues: 1) normal data from the user, 2) expedited data from the user, and 3) messages from the network. If the transport process then wants to give top priority to network messages, middle priority to expedited user messages, and lowest priority to normal user messages, all that is required is receipt of IPC messages on the highest priority queue until no more messages are available. Then the receiver moves to the next lower in priority and so on. More sophistication is possible by setting limits upon the number of consecutive messages received from each queue and/or varying the order in which each queue is examined.

It is easy to see how a round-robin scheduling discipline could be implemented using this form of IPC.

Approaches to IPC can be placed into one of three classes: 1) shared memory, 2) memory-memory copying, and 3) input/output channel copying. Shared memory is the most desirable of the three classes because the amount of data movement is kept to a minimum. To pass IPC messages using shared memory, the sender builds a small message referencing a potentially large amount of user data. The small message is then either copied from the sender's process space to the receiver's process space or the small message is mapped from one process space to another using techniques specific to the operating system and hardware involved. These approaches to shared memory are equivalent since the amount of data movement is kept to a minimum. The price to be paid for using this approach is due to the synchronization of access to the shared memory. This type of sharing is well understood, and several efficient and simple techniques exist to manage the sharing.

Memory-memory copying is an approach that has been commonly used for IPC in UNIX operating system implementations. To pass an IPC message under UNIX data is copied from the sender's buffer to a kernel buffer and then from a kernel buffer to the receiver's buffer. Thus two copy operations are required for each IPC message. Other methods might only involve a single copy operation. Also note that if one of the processes involved is the transport protocol implemented in the kernel, the IPC message must only be copied once. The main disadvantage of this approach is inefficiency. The major advantage is simplicity.

When the processes that must exchange messages reside on physically separate computer systems (e.g., a host and front end), an input/output channel of some type must be used to support the IPC. In such a case, the problem is similar to that of the general problem of a transport protocol. The sender must provide his IPC message to some standard operating system output mechanism from where it will be transmitted via some physical medium to the receiver's operating system. The receiver's operating system will then pass the message on to the receiving process via some standard operating system input mechanism. This set of procedures can vary greatly in efficiency and complexity depending upon the operating systems and hardware involved. Usually this approach to IPC is used only when the circumstances require it.

3.4 Interface to real networks.

Implementations of the class 4 transport protocol have been operated over a wide variety of networks including: 1) ARPANET, 2) X.25 networks, 3) satellite channels, 4) CSMA/CD local area networks, 5) token bus local area networks, and 6) token ring local area networks. This section briefly describes known instances of each use

of class 4 transport and provides some quantitative evaluation of the performance expectations for transport over each network type.

3.4.1 Issues.

The interface of the transport entity to the network service in general will be realized in a different way from the user interface. The network service processor is often separate from the host CPU, connected to it by a bus, direct memory access (DMA), or other link. A typical way to access the network service is by means of a device driver. The transfer of data across the interface in this instance would be by buffer-copying. The use of double-buffering reduces some of the complexity of flow control, which is usually accomplished by examining the capacity of the target buffer. If the transport processor and the network processor are distinct and connected by a bus or external link, the network access may be more complicated since copying will take place across the bus or link rather than across the memory board. In any case, the network service primitives, as they appear in the formal description and IS 8073 must be carefully correlated to the actual access scheme, so that the semantics of the primitives is preserved. One way to do this is to create a library of routines, each of which corresponds to one of the service primitives. Each routine is responsible for sending the proper signal to the network interface unit, whether this communication is directly, as on a bus, or indirectly via a device driver. In the case of a connectionless network service, there is only one primitive, the N_DATA_request (or N_UNIT_DATA_request), which has to be realized.

In the formal description, flow control to the NSAP is controlled by by a Slave module, which exerts the "backpressure" on the TPM if its internal queue gets too long. Incoming flow, however, is controlled in much the same way as the flow to the transport user is controlled. The implementor is reminded that the formal description of the flow control is specified for completeness and not as an implementation guide. Thus, an implementation should depend upon actual interfaces in the operating environment to realize necessary functions.

3.4.2 Instances of operation.

3.4.2.1 ARPANET

An early implementation of the class 4 transport protocol was developed by the NBS as a basis for conformance tests [NBS83]. This implementation was used over the ARPANET to communicate between NBS, BBN, and DCA. The early NBS implementation was executed on a PDP-11/70. A later revision of the NBS implementation has been moved to a VAX-11/750 and VAX-11/7;80. The Norwegian Telecommunication Administration (NTA) has implemented class 4 transport for the UNIX BSD 4.2 operating system to run on a VAX [NTA84]. A later NTA implementation runs on a Sun 2-120 workstation. The University of

Wisconsin has also implemented the class 4 transport protocol on a VAX-11/750 [BRI85]. The Wisconsin implementation is embedded in the BSD 4.2 UNIX kernel. For most of these implementations class 4 transport runs above the DOD IP and below DOD application protocols.

3.4.2.2 X.25 networks

The NBS implementations have been used over Telenet, an X.25 public data network (PDN). The heaviest use has been testing of class 4 transport between the NBS and several remotely located vendors, in preparation for a demonstration at the 1984 National Computing Conference and the 1985 Autofact demonstration. Several approaches to implementation were seen in the vendors' systems, including ones similar to those discussed in Part 6.2. At the Autofact demonstration many vendors operated class 4 transport and the ISO internetwork protocol across an internetwork of CSMA/CD and token bus local networks and Accunet, an AT&T X.25 public data network.

3.4.2.3 Satellite channels.

The COMSAT Laboratories have implemented class 4 transport for operation over point-to-point satellite channels with data rates up to 1.544 Mbps [CHO85]. This implementation has been used for experiments between the NBS and COMSAT. As a result of these experiments several improvements have been made to the class 4 transport specification within the international standards arena (both ISO and CCITT). The COMSAT implementation runs under a proprietary multiprocessing operating system known as COSMOS. The hardware base includes multiple Motorola 68010 CPUs with local memory and Multibus shared memory for data messages.

3.4.2.4 CSMA/CD networks.

The CSMA/CD network as defined by the IEEE 802.3 standard is the most popular network over which the class 4 transport has been implemented. Implementations of transport over CSMA/CD networks have been demonstrated by: AT&T, Charles River Data Systems, Computervision, DEC, Hewlitt-Packard, ICL, Intel, Intergraph, NCR and SUN. Most of these were demonstrated at the 1984 National Computer Conference [MIL85b] and again at the 1985 Autofact Conference. Several of these vendors are now delivering products based on the demonstration software.

3.4.2.5 Token bus networks.

Due to the establishment of class 4 transport as a mandatory protocol within the General Motor's manufacturing automation protocol (MAP), many implementations have been demonstrated operating over a token bus network as defined by the IEEE 802.4 standard. Most past implementations relied upon a Concord Data Systems token interface module (TIM) to gain access to the 5 Mbps broadband 802.4 service.

Several vendors have recently announced boards supporting a 10 Mbps broadband 802.4 service. The newer boards plug directly into computer system buses while the TIM's are accessed across a high level data link control (HDLC) serial channel. Vendors demonstrating class 4 transport over IEEE 802.4 networks include Allen-Bradley, AT&T, DEC, Gould, Hewlett-Packard, Honeywell, IBM, Intel, Motorola, NCR and Siemens.

3.4.2.6 Token ring networks.

The class 4 transport implementations by the University of Wisconsin and by the NTA run over a 10 Mbps token ring network in addition to ARPANET. The ring used is from Proteon rather than the recently finished IEEE 802.5 standard.

3.4.3 Performance expectations.

Performance research regarding the class 4 transport protocol has been limited. Some work has been done at the University of Wisconsin, at NTA, at Intel, at COMSAT, and at the NBS. The material presented below draws from this limited body of research to provide an implementor with some quantitative feeling for the performance that can be expected from class 4 transport implementations using different network types. More detail is available from several published reports [NTA84, BRI85, INT85, MIL85b, COL85]. Some of the results reported derive from actual measurements while other results arise from simulation. This distinction is clearly noted.

3.4.3.1 Throughput.

Several live experiments have been conducted to determine the throughput possible with implementations of class 4 transport. Achievable throughput depends upon many factors including: 1) CPU capabilities, 2) use or non-use of transport checksum, 3) IPC mechanism, 4) buffer management technique, 5) receiver resequencing, 6) network error properties, 7) transport flow control, 8) network congestion and 9) TPDU size. Some of these are specifically discussed elsewhere in this document. The reader must keep in mind these issues when interpreting the throughput measures presented here.

The University of Wisconsin implemented class 4 transport in the UNIX kernel for a VAX-11/750 with the express purpose of measuring the achievable throughput. Throughputs observed over the ARPANET ranged between 10.4 Kbps and 14.4 Kbps. On an unloaded Proteon ring local network, observed throughput with checksum ranged between 280 Kbps and 560 Kbps. Without checksum, throughput ranged between 384 Kbps and 1 Mbps.

The COMSAT Laboratories implemented class 4 transport under a proprietary multiprocessor operating system for a multiprocessor

68010 hardware architecture. The transport implementation executed on one 68010 while the traffic generator and link drivers executed on a second 68010. All user messages were created in a global shared memory and were copied only for transmission on the satellite link. Throughputs as high as 1.4 Mbps were observed without transport checksumming while up to 535 Kbps could be achieved when transport checksums were used. Note that when the 1.4 Mbps was achieved the transport CPU was idle 20% of the time (i.e., the 1.544 Mbps satellite link was the bottleneck). Thus, the transport implementation used here could probably achieve around 1.9 Mbps user throughput with the experiment parameters remaining unchanged. Higher throughputs are possible by increasing the TPDU size; however, larger messages stand an increased chance of damage during transmission.

Intel has implemented a class 4 transport product for operation over a CSMA/CD local network (iNA-960 running on the iSBC 186/51 or iSXM 552). Intel has measured throughputs achieved with this combination and has published the results in a technical analysis comparing iNA-960 performance on the 186/51 with iNA-960 on the 552. The CPU used to run transport was a 6 MHz 80186. An 82586 co-processor was used to handle the medium access control. Throughputs measured ranged between 360 Kbps and 1.32 Mbps, depending on the parameter values used.

Simulation of class 4 transport via a model developed at the NBS has been used to predict the performance of the COMSAT implementation and is now being used to predict the performance of a three processor architecture that includes an 8 MHz host connected to an 8 MHz front end over a system bus. The third processor provides medium access control for the specific local networks being modeled. Early model results predict throughputs over an unloaded CSMA/CD local network of up to 1.8 Mbps. The same system modeled over a token bus local network with the same transport parameters yields throughput estimates of up to 1.6 Mbps. The token bus technology, however, permits larger message sizes than CSMA/CD does. When TPDUs of 5120 bytes are used, throughput on the token bus network is predicted to reach 4.3 Mbps.

3.4.3.2 Delay.

The one-way delay between sending transport user and receiving transport user is determined by a complex set of factors. Readers should also note that, in general, this is a difficult measure to make and little work has been done to date with respect to expected one-way delays with class 4 transport implementations. In this section a tutorial is given to explain the factors that determine the one-way delay to be expected by a transport user. Delay experiments performed by Intel are reported [INT85], as well as some simulation experiments conducted by the NBS [MIL85a].

The transport user can generally expect one-way delays to be determined by the following equation.

$$D = TS + ND + TR + [IS] + [IR] \qquad (1)$$

where:

[.] means the enclosed quantity may be 0

D is the one-way transport user delay,

TS is the transport data send processing time,

IS is the internet datagram send processing time,

ND is the network delay,

IR is the internet datagram receive processing time, and

TR is the transport data receive processing time.

Although no performance measurements are available for the ISO internetwork protocol (ISO IP), the ISO IP is so similar to the DOD IP that processing times associated with sending and receiving datagrams should be the about the same for both IPs. Thus, the IS and IR terms given above are ignored from this point on in the discussion. Note that many of these factors vary depending upon the application traffic pattern and loads seen by a transport implementation. In the following discussion, the transport traffic is assumed to be a single message.

The value for TS depends upon the CPU used, the IPC mechanism, the use or non-use of checksum, the size of the user message and the size of TPDUs, the buffer management scheme in use, and the method chosen for timer management. Checksum processing times have been observed that include 3.9 us per octet for a VAX-11/750, 7.5 us per octet on a Motorola 68010, and 6 us per octet on an Intel 80186. The class 4 transport checksum algorithm has considerable effect on achievable performance. This is discussed further in Part 7. Typical values for TS, excluding the processing due to the checksum, are about 4 ms for CPUs such as the Motorola 68010 and the Intel 80186. For 1024 octet TPDUs, checksum calculation can increase the TS value to about 12 ms.

The value of TR depends upon similar details as TS. An additional consideration is whether or not the receiver caches (buffers) out of order TPDUs. If so, the TR will be higher when no packets are lost (because of the overhead incurred by the resequencing logic). Also,

when packets are lost, TR can appear to increase due to transport resequencing delay. When out of order packets are not cached, lost packets increase D because each unacknowledged packet must be retransmitted (and then only after a delay waiting for the retransmission timer to expire). These details are not taken into account in equation 1. Typical TR values that can be expected with non-caching implementations on Motorola 68010 and Intel 80186 CPUs are approximately 3 to 3.5 ms. When transport checksumming is used on these CPUs, TR becomes about 11 ms for 1024 byte TPDUs.

The value of ND is highly variable, depending on the specific network technology in use and on the conditions in that network. In general, ND can be defined by the following equation.

$$ND = NQ + MA + TX + PD + TQ \quad (2)$$

where:

NQ is network queuing delay,

MA is medium access delay,

TX is message transmission time,

PD is network propagation delay, and

TQ is transport receive queuing delay.

Each term of the equation is discussed in the following paragraphs.

Network queuing delay (NQ) is the time that a TPDU waits on a network transmit queue until that TPDU is the first in line for transmission. NQ depends on the size of the network transmit queue, the rate at which the queue is emptied, and the number of TPDUs already on the queue. The size of the transmit queue is usually an implementation parameter and is generally at least two messages. The rate at which the queue empties depends upon MA and TX (see the discussion below). The number of TPDUs already on the queue is determined by the traffic intensity (ratio of mean arrival rate to mean service rate). As an example, consider an 8 Kbps point-to-point link serving an eight message queue that contains 4 messages with an average size of 200 bytes per message. The next message to be placed into the transmit queue would experience an NQ of 800 ms (i.e., 4 messages times 200 ms). In this example, MA is zero. These basic facts permit the computation of NQ for particular environments. Note that if the network send queue is full, back pressure flow control will force TPDUs to queue in transport transmit buffers and cause TS to appear to increase by the amount of the transport queuing delay. This condition depends on application traffic patterns but is ignored for

the purpose of this discussion.

The value of MA depends upon the network access method and on the network congestion or load. For a point-to-point link MA is zero. For CSMA/CD networks MA depends upon the load, the number of stations, the arrival pattern, and the propagation delay. For CSMA/CD networks MA has values that typically range from zero (no load) up to about 3 ms (80% loads). Note that the value of MA as seen by individual stations on a CSMA/CD network is predicted (by NBS simulation studies) to be as high as 27 ms under 70% loads. Thus, depending upon the traffic patterns, individual stations may see an average MA value that is much greater than the average MA value for the network as a whole. On token bus networks MA is determined by the token rotation time (TRT) which depends upon the load, the number of stations, the arrival pattern, the propagation delay, and the values of the token holding time and target rotation times at each station.

For small networks of 12 stations with a propagation delay of 8 ns, NBS simulation studies predict TRT values of about 1 ms for zero load and 4.5 ms for 70% loads for 200 byte messages arriving with exponential arrival distribution. Traffic patterns also appear to be an important determinant of target rotation time. When a pair of stations performs a continuous file transfer, average TRT for the simulated network is predicted to be 3 ms for zero background load and 12.5 ms for 70% background load (total network load of 85%).

The message size and the network transmission speed directly determine TX. Typical transmission speeds include 5 and 10 Mbps for standard local networks; 64 Kbps, 384 Kbps, or 1.544 Mbps for point-to-point satellite channels; and 9.6 Kbps or 56 Kbps for public data network access links.

The properties of the network in use determine the values of PD. On an IEEE 802.3 network, PD is limited to 25.6 us. For IEEE 802.4 networks, the signal is propagated up-link to a head end and then down-link from the head end. Propagation delay in these networks depends on the distance of the source and destination stations from the head end and on the head end latency. Because the maximum network length is much greater than with IEEE 802.3 networks, the PD values can also be much greater. The IEEE 802.4 standard requires that a network provider give a value for the maximum transmission path delay. For satellite channels PD is typically between 280 and 330 ms. For the ARPANET, PD depends upon the number of hops that a message makes between source and destination nodes. The NBS and NTIA measured ARPANET PD average values of about 190 ms [NTI85]. In the ARPA internet system the PD is quite variable, depending on the number of internet gateway hops and the PD values of any intervening networks (possibly containing satellite channels). In experiments on an internetwork containing a a satellite link to Korea, it was determined by David Mills [RFC85] that internet PD values could range from 19 ms to 1500 ms. Thus, PD values ranging from 300 to 600 ms

can be considered as typical for ARPANET internetwork operation.

The amount of time a TPDU waits in the network receive queue before being processed by the receiving transport is represented by TQ, similar to NQ in that the value of TQ depends upon the size of the queue, the number of TPDUs already in the queue, and the rate at which the queue is emptied by transport.

Often the user delay D will be dominated by one of the components. On a satellite channel the principal component of D is PD, which implies that ND is a principal component by equation (2). On an unloaded LAN, TS and TR might contribute most to D. On a highly loaded LAN, MA may cause NQ to rise, again implying that ND is a major factor in determining D.

Some one-way delay measures have been made by Intel for the iNA-960 product running on a 6 MHz 80186. For an unloaded 10 Mbps CSMA/CD network the Intel measures show delays as low as 22 ms. The NBS has done some simulations of class 4 transport over 10 Mbps CSMA/CD and token bus networks. These (unvalidated) predictions show one-way delays as low as 6 ms on unloaded LANs and as high as 372 ms on CSMA/CD LANs with 70% load.

3.4.3.3 Response time.

Determination of transport user response time (i.e., two-way delay) depends upon many of the same factors discussed above for one-way delay. In fact, response time can be represented by equation 3 as shown below.

$$R = 2D + AS + AR \qquad (3)$$

where:

R is transport user response time,

D is one-way transport user delay,

AS is acknowledgement send processing time, and

AR is acknowledgement receive processing time.

D has been explained above. AS and AR deal with the acknowledgement sent by transport in response to the TPDU that embodies the user request.

AS is simply the amount of time that the receiving transport must spend to generate an AK TPDU. Typical times for this function are about 2 to 3 ms on processors such as the Motorola 68010 and the Intel 80186. Of course the actual time required depends upon factors such as those explained for TS above.

The amount of time, AR, that the sending transport must spend to process a received AK TPDU. Determination of the actual time required depends upon factors previously described. Note that for AR and AS, processing when the checksum is included takes somewhat longer. However, AK TPDUs are usually between 10 and 20 octets in length and therefore the increased time due to checksum processing is much less than for DT TPDUs.

No class 4 transport user response time measures are available; however, some simulations have been done at the NBS. These predictions are based upon implementation strategies that have been used by commercial vendors in building microprocessor-based class 4 transport products. Average response times of about 21 ms on an unloaded 10 Mbps token bus network, 25 ms with 70% loading, were predicted by the simulations. On a 10 Mbps CSMA/CD network, the simulations predict response times of about 17 ms for no load and 54 ms for a 70% load.

3.5 Error and status reporting.

Although the abstract service definition for the transport protocol specifies a set of services to be offered, the actual set of services provided by an implementation need not be limited to these. In particular, local status and error information can be provided as a confirmed service (request/response) and as an asynchronous "interrupt" (indication). One use for this service is to allow users to query the transport entity about the status of their connections. An example of information that could be returned from the entity is:

- connection state
- current send sequence number
- current receive and transmit credit windows
- transport/network interface status
- number of retransmissions
- number of DTs and AKs sent and received
- current timer values

Another use for the local status and error reporting service is for administration purposes. Using the service, an administrator can gather information such as described above for each open connection. In addition, statistics concerning the transport entity as a whole can be obtained, such as number of transport connections open, average number of connections open over a given reporting period, buffer use statistics, and total number of retransmitted DT TPDUs. The administrator might also be given the authority to cancel connections, restart the entity, or manually set timer values.

4 Entity resource management.

4.1 CPU management.

The formal description has implicit scheduling of TPM modules, due to the semantics of the Estelle structuring principles. However, the implementor should not depend on this scheduling to obtain optimal behavior, since, as stated in Part 1, the structures in the formal description were imposed for purposes other than operational efficiency.

Whether by design or by default, every implementation of the transport protocol embodies some decision about allocating the CPU resource among transport connections. The resource may be monolithic, i.e. a single CPU, or it may be distributed, as in the example design given in Part 2.3. In the former, there are two simple techniques for apportioning CPU processing time among transport connections. The first of these, first-come/first-served, consists of the transport entity handling user service requests in the order in which they arrive. No attempt is made to prevent one transport connection from using an inordinate amount of the CPU.

The second simple technique is round-robin scheduling of connections. Under this method, each transport connection is serviced in turn. For each connection, transport processes one user service request, if there is one present at the interface, before proceeding to the next connection.

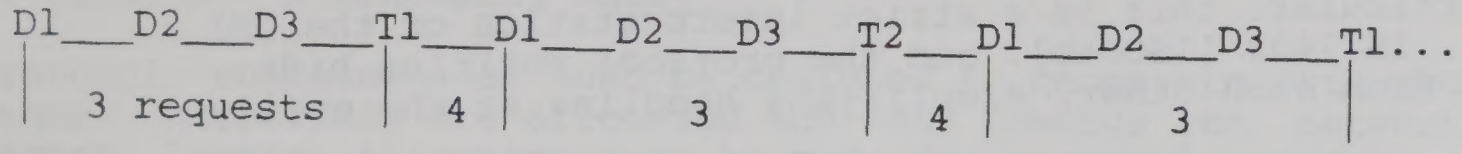
The quality of service parameters provided in the connection request can be used to provide a finer-grained strategy for managing the CPU. The CPU could be allocated to connections requiring low delay more often while those requiring high throughput would be served less often but for longer periods (i.e., several connections requiring high throughput might be serviced in a concurrent cluster).

For example, in the service sequence below, let "T" represent m > 0 service requests, each requiring high throughput, let "D" represent one service request requiring low delay and let the suffix n = 1,2,3 represent a connection identifier, unique only within a particular service requirement type (T,D). Thus T1 represents a set of service requests for connection 1 of the service requirement type T, and D1 represents a service set (with one member) for connection 1 of service requirement type D.

D1___D2___D3___T1___D1___D2___D3___T2___D1___D2___D3___T1...

If m = 4 in this service sequence, then service set D1 will get worst-case service once every seventh service request processed. Service set T1 receives service on its four requests only once in

fourteen requests processed.



This means that the CPU is allocated to T1 29% (4/14) of the available time, whereas D1 obtains service 14% (1/7) of the time, assuming processing requirements for all service requests to be equal. Now assume that, on average, there is a service request arriving for one out of three of the service requirement type D connections. The CPU is then allocated to the T type 40% (4/10) while the D type is allocated 10% (1/10).

4.2 Buffer management.

Buffers are used as temporary storage areas for data on its way to or arriving from the network. Decisions must be made about buffer management in two areas. The first is the overall strategy for managing buffers in a multi-layered protocol environment. The second is specifically how to allocate buffers within the transport entity.

In the formal description no details of buffer strategy are given, since such strategy depends so heavily on the implementation environment. Only a general mechanism is discussed in the formal description for allocating receive credit to a transport connection, without any expression as to how this resource is managed.

Good buffer management should correlate to the traffic presented by the applications using the transport service. This traffic has implications as well for the performance of the protocol. At present, the relationship of buffer strategy to optimal service for a given traffic distribution is not well understood. Some work has been done, however, and the reader is referred to the work of Jeffery Spirn [SPI82, SPI83] and to the experiment plan for research by the NBS [HEA85] on the effect of application traffic patterns on the performance of Class 4 transport.

4.2.1 Overall buffer strategy.

Three schemes for management of buffers in a multilayered environment are described here. These represent a spectrum of possibilities available to the implementor. The first of these is a strictly layered approach in which each entity in the protocol hierarchy, as a process, manages its own pool of buffers independently of entities at other layers. One advantage of this approach is simplicity; it is not necessary for an entity to coordinate buffer usage with a resource manager which is serving the needs of numerous protocol entities. Another advantage is modularity. The interface presented to entities in other layers is

well defined; protocol service requests and responses are passed between layers by value (copying) versus by reference (pointer copying). In particular, this is a strict interpretation of the OSI reference model, IS 7498 [ISO84b], and the protocol entities hide message details from each other, simplifying handling at the entity interfaces.

The single disadvantage to a strictly layered scheme derives from the value-passing nature of the interface. Each time protocol data and control information is passed from one layer to another it must be copied from one layer's buffers to those of another layer. Copying between layers in a multi-layered environment is expensive and imposes a severe penalty on the performance of the communications system, as well as the computer system on which it is running as a whole.

The second scheme for managing buffers among multiple protocol layers is buffer sharing. In this approach, buffers are a shared resource among multiple protocol entities; protocol data and control information contained in the buffers is exchanged by passing a buffer pointer, or reference, rather than the values as in the strictly layered approach described above. The advantage to passing buffers by reference is that only a small amount of information, the buffer pointer, is copied from layer to layer. The resulting performance is much better than that of the strictly layered approach.

There are several requirements that must be met to implement buffer sharing. First, the host system architecture must allow memory sharing among protocol entities that are sharing the buffers. This can be achieved in a variety of ways: multiple protocol entities may be implemented as one process, all sharing the same process space (e.g., kernel space), or the host system architecture may allow processes to map portions of their address space to common buffer areas at some known location in physical memory.

A buffer manager is another requirement for implementing shared buffers. The buffer manager has the responsibility of providing buffers to protocol entities when needed from a list of free buffers and recycling used buffers back into the free list. The pool may consist of one or more lists, depending on the level of control desired. For example, there could be separate lists of buffers for outgoing and incoming messages.

The protocol entities must be implemented in such a way as to cooperate with the buffer manager. While this appears to be an obvious condition, it has important implications for the strategy used by implementors to develop the communications system. This cooperation can be described as follows: an entity at layer N requests and is allocated a buffer by the manager; each such buffer

is returned to the manager by some entity at layer N - k (outgoing data) or N + k (incoming data).

Protocol entities also must be designed to cooperate with each other. As buffers are allocated and sent towards the network from higher layers, allowance must be made for protocol control information to be added at lower layers. This usually means allocating oversized buffers to allow space for headers to be prepended at lower layers. Similarly, as buffers move upward from the network, each protocol entity processes its headers before passing the buffer on. These manipulations can be handled by managing pointers into the buffer header space.

In their pure forms, both strictly layered and shared buffer schemes are not practical. In the former, there is a performance penalty for copying buffers. On the other hand, it is not practical to implement buffers that are shared by entities in all layers of the protocol hierarchy: the lower protocol layers (OSI layers 1 - 4) have essentially static buffer requirements, whereas the upper protocol layers (OSI layers 5 - 7) tend to be dynamic in their buffer requirements. That is, several different applications may be running concurrently, with buffer requirements varying as the set of applications varies. However, at the transport layer, this latter variation is not visible and variations in buffer requirements will depend more on service quality considerations than on the specific nature of the applications being served. This suggests a hybrid scheme in which the entities in OSI layers 1 - 4 share buffers while the entities in each of the OSI layers 5 - 7 share in a buffer pool associated with each layer. This approach provides most of the efficiency of a pure shared buffer scheme and allows for simple, modular interfaces where they are most appropriate.

4.2.2 Buffer management in the transport entity.

Buffers are allocated in the transport entity for two purposes: sending and receiving data. For sending data, the decision of how much buffer space to allocate is relatively simple; enough space should be allocated for outgoing data to hold the maximum number of data messages that the entity will have outstanding (i.e., sent but unacknowledged) at any time. The send buffer space is determined by one of two values, whichever is lower: the send credit received from the receiving transport entity, or a maximum value imposed by the local implementation, based on such factors as overall buffer capacity.

The allocation of receive buffers is a more interesting problem because it is directly related to the credit value transmitted the peer transport entity in CR (or CC) and AK TPDUs. If the total credit offered to the peer entity exceeds the total available buffer space and credit reduction is not implemented, deadlock may occur, causing termination of one or more transport connections. For

the purposes of this discussion, offered credit is assumed to be equivalent to available buffer space.

The simplest scheme for receive buffer allocation is allocation of a fixed amount per transport connection. This amount is allocated regardless of how the connection is to be used. This scheme is fair in that all connections are treated equally. The implementation approach in Part 2.3, in which each transport connection is handled by a physically separate processor, obviously could use this scheme, since the allocation would be in the form of memory chips assigned by the system designer when the system is built.

A more flexible method of allocating receive buffer space is based on the connection quality of service (QOS) requested by the user. For instance, a QOS indicating high throughput would be given more send and receive buffer space than one a QOS indicating low delay. Similarly, connection priority can be used to determine send and receive buffer allocation, with important (i.e., high priority) connections allocated more buffer space.

A slightly more complex scheme is to apportion send and receive buffer space using both QOS and priority. For each connection, QOS indicates a general category of operation (e.g., high throughput or low delay). Within the general category, priority determines the specific amount of buffer space allocated from a range of possible values. The general categories may well overlap, resulting, for example, in a high priority connection with low throughput requirements being allocated more buffer space than low priority connection requiring a high throughput.

5 Management of Transport service endpoints.

As mentioned in Part 1.2.1.1, a transport entity needs some way of referencing a transport connection endpoint within the end system: a TCEP_id. There are several factors influencing the management of TCEP_ids:

1) IPC mechanism between the transport entity and the session entity (Part 3.3);

2) transport entity resources and resource management (Part 4);

3) number of distinct TSAPs supported by the entity (Part 1.2.2.1); and

4) user process rendezvous mechanism (the means by which session processes identify themselves to the transport entity, at a given TSAP, for association with a transport connection);

The IPC mechanism and the user process rendezvous mechanism have more direct influence than the other two factors on how the TCEP_id

management is implemented.

The number of TCEP_ids available should reflect the resources that are available to the transport entity, since each TCEP_id in use represents a potential transport connection. The formal description assumes that there is a function in the TPE which can decide, on the basis of current resource availability, whether or not to issue a TCEP_id for any connection request received. If the TCEP_id is issued, then resources are allocated for the connection endpoint. However, there is a somewhat different problem for the users of transport. Here, the transport entity must somehow inform the session entity as to the TCEP_ids available at a given TSAP.

In the formal description, a T-CONNECT-request is permitted to enter at any TSAP/TCEP_id. A function in the TPE considers whether or not resources are availble to support the requested connection. There is also a function which checks to see if a TSAP/TCEP_id is busy by seeing if there is a TPM allocated to it. But this function is not useful to the session entity which does not have access to the transport entity's operations. This description of the procedure is clearly too loose for an implementation.

One solution to this problem is to provide a new (abstract) service, T-REGISTER, locally, at the interface between transport and session.

Primitives		Parameters
T-REGISTER	request	Session process identifier
T-REGISTER	indication	Transport endpoint identifier, Session process identifier
T-REGISTER	refusal	Session process identifier

This service is used as follows:

1) A session process is identified to the transport entity by a T-REGISTER-request event. If a TCEP_id is available, the transport entity selects a TCEP_id and places it into a table corresponding to the TSAP at which the T-REGISTER-request event occurred, along with the session process identifier. The TCEP_id and the session process identifier are then transmitted to the session entity by means of the T-REGISTER-indication event. If no TCEP_id is available, then a T-REGISTER-refusal event carrying the session process identifier is returned. At any time that an assigned TCEP_id is not associated with an active transport connection process (allocated TPM), the transport entity can issue a T-REGISTER-

refusal to the session entity to indicate, for example, that resources are no longer available to support a connection, since TC resources are not allocated at registration time.

2) If the session entity is to initiate the transport connection, it issues a T-CONNECT-request with the TCEP_id as a parameter. (Note that this procedure is at a slight variance to the procedure in N3756, which specifies no such parameter, due to the requirement of alignment of the formal description with the service description of transport and the definition of the session protocol.) If the session entity is expecting a connection request from a remote peer at this TSAP, then the transport does nothing with the TCEP_id until a CR TPDU addressed to the TSAP arrives. When such a CR TPDU arrives, the transport entity issues a T-CONNECT-indication to the session entity with a TCEP_id as a parameter. As a management aid, the table entry for the TCEP_id can be marked "busy" when the TCEP_id is associated with an allocated TPM.

3) If a CR TPDU is received and no TCEP_id is in the table for the TSAP addressed, then the transport selects a TCEP_id, includes it as a parameter in the T-CONNECT-indication sent to the session entity, and places it in the table. The T-CONNECT-response returned by the session entity will carry the TCEP_id and the session process identifier. If the session process identifier is already in the table, the new one is discarded; otherwise it is placed into the table. This procedure is also followed if the table has entries but they are all marked busy or are empty. If the table is full and all entries ar marked busy, then the transport entity transmits a DR TPDU to the peer transport entity to indicate that the connection cannot be made. Note that the transport entity can disable a TSAP by marking all its table entries busy.

The realization of the T-REGISTER service will depend on the IPC mechanisms available between the transport and session entities. The problem of user process rendezvous is solved in general by the T-REGISTER service, which is based on a solution proposed by Michael Chernik of the NBS [CHK85].

6 Management of Network service endpoints in Transport.

6.1 Endpoint identification.

The identification of endpoints at an NSAP is different from that for the TSAP. The nature of the services at distinct TSAPs is fundamentally the same, although the quality could vary, as a local

choice. However, it is possible for distinct NSAPs to represent access to essentially different network services. For example, one NSAP may provide access to a connectionless network service by means of an internetwork protocol. Another NSAP may provide access to a connection-oriented service, for use in communicating on a local subnetwork. It is also possible to have several distinct NSAPs on the same subnetwork, each of which provides some service features of local interest that distinguishes it from the other NSAPs.

A transport entity accessing an X.25 service could use the logical channel numbers for the virtual circuits as NCEP_ids. An NSAP providing access only to a permanent virtual circuit would need only a single NCEP_id to multiplex the transport connections. Similarly, a CSMA/CD network would need only a single NCEP_id, although the network is connectionless.

6.2 Management issues.

The Class 4 transport protocol has been succesfully operated over both connectionless and connection-oriented network services. In both modes of operation there exists some information about the network service that a transport implementation could make use of to enhance performance. For example, knowledge of expected delay to a destination would permit optimal selection of retransmission timer value for a connection instance. The information that transport implementations could use and the mechanisms for obtaining and managing that information are, as a group, not well understood. Projects are underway within ISO committees to address the management of OSI as an architecture and the management of the transport layer as a layer.

For operation of the Class 4 transport protocol over connection-oriented network service several issues must be addressed including:

a. When should a new network connection be opened to support a transport connection (versus multiplexing)?

b. When a network connection is no longer being used by any transport connection, should the network connection be closed or remain open awaiting a new transport connection?

c. When a network connection is aborted, how should the peer transport entities that were using the connection cooperate to re-establish it? If splitting is not to be used, how can this re-establishment be achieved such that one and only one network connection results?

The Class 4 transport specification permits a transport entity to multiplex several transport connections (TCs) over a single network

connection (NC) and to split a single TC across several NCs. The implementor must decide whether to support these options and, if so, how. Even when the implementor decides never to initiate splitting or multiplexing the transport entity must be prepared to accept this behavior from other transport implementations. When multiplexing is used TPDUs from multiple TCs can be concatenated into a single network service data unit (NSDU). Therefore, damage to an NSDU may effect several TCs. In general, Class 2 connections should not be multiplexed with Class 4 connections. The reason for this is that if the error rate on the network connection is high enough that the error recovery capability of Class 4 is needed, then it is too high for Class 2 operation. The deciding criterion is the tolerance of the user for frequent disconnection and data errors.

Several issues in splitting must be considered:

1) maximum number of NCs that can be assigned to a given TC;

2) minimum number of NCs required by a TC to maintain the "quality of service" expected (default of 1);

3) when to split;

4) inactivity control;

5) assignment of received TPDU to TC; and

6) notification to TC of NC status (assigned, dissociated, etc).

All of these except 3) are covered in the formal description. The methods used in the formal description need not be used explicitly, but they suggest approaches to implementation.

To support the possibility of multiplexing and splitting the implementor must provide a common function below the TC state machines that maps a set of TCs to a set of NCs. The formal description provides a general means of doing this, requiring mainly implementation environment details to complete the mechanism. Decisions about when network connections are to be opened or closed can be made locally using local decision criteria. Factors that may effect the decision include costs of establishing an NC, costs of maintaining an open NC with little traffic flowing, and estimates of the probability of data flow between the source node and known destinations. Management of this type is feasible when a priori knowledge exists but is very difficult when a need exists to adapt to dynamic traffic patterns and/or fluctuating network charging mechanisms.

To handle the issue of re-establishment of the NC after failure, the ISO has proposed an addendum N3279 [ISO85c] to the basic transport standard describing a network connection management subprotocol

(NCMS) to be used in conjunction with the transport protocol.

7 Enhanced checksum algorithm.

7.1 Effect of checksum on transport performance.

Performance experiments with Class 4 transport at the NBS have revealed that straightforward implementation of the Fletcher checksum using the algorithm recommended in the ISO transport standard leads to severe reduction of transport throughput. Early modeling indicated throughput drops of as much as 66% when using the checksum. Work by Anastase Nakassis [NAK85] of the NBS led to several improved implementations. The performance degradation due to checksum is now in the range of 40-55%, when using the improved implementations.

It is possible that transport may be used over a network that does not provide error detection. In such a case the transport checksum is necessary to ensure data integrity. In many instances, the underlying subnetwork provides some error checking mechanism. The HDLC frame check sequence as used by X.25, IEEE 802.3 and 802.4 rely on a 32 bit cyclic redundancy check and satellite link hardware frequently provides the HDLC frame check sequence. However, these are all link or physical layer error detection mechanisms which operate only point-to-point and not end-to-end as the transport checksum does. Some links provide error recovery while other links simply discard damaged messages. If adequate error recovery is provided, then the transport checksum is extra overhead, since transport will detect when the link mechanism has discarded a message and will retransmit the message. Even when the IP fragments the TPDU, the receiving IP will discover a hole in the reassembly buffer and discard the partially assembled datagram (i.e., TPDU). Transport will detect this missing TPDU and recover by means of the retransmission mechanism.

7.2 Enhanced algorithm.

The Fletcher checksum algorithm given in an annex to IS 8073 is not part of the standard, and is included in the annex as a suggestion to implementors. This was done so that as improvements or new algorithms came along, they could be incorporated without the necessity to change the standard.

Nakassis has provided three ways of coding the algorithm, shown below, to provide implementors with insight rather than universally transportable code. One version uses a high order language (C). A second version uses C and VAX assembler, while a third uses only VAX assembler. In all the versions, the constant MODX appears. This represents the maximum number of sums that can be taken without experiencing overflow. This constant depends on the processor's word size and the arithmetic mode, as follows:

Choose n such that

```
(n+1)*(254 + 255*n/2) <= 2**N - 1
```

where N is the number of usable bits for signed (unsigned) arithmetic. Nakassis shows [NAK85] that it is sufficient to take

```
n <= sqrt( 2*(2**N - 1)/255 )
```

and that n = sqrt(2*(2**N - 1)/255) - 2 generally yields usable values. The constant MODX then is taken to be n.

Some typical values for MODX are given in the following table.

BITS/WORD	MODX	ARITHMETIC
15	14	signed
16	21	unsigned
31	4102	signed
32	5802	unsigned

This constant is used to reduce the number of times mod 255 addition is invoked, by way of speeding up the algorithm.

It should be noted that it is also possible to implement the checksum in separate hardware. However, because of the placement of the checksum within the TPDU header rather than at the end of the TPDU, implementing this with registers and an adder will require significant associated logic to access and process each octet of the TPDU and to move the checksum octets in to the proper positions in the TPDU. An alternative to designing this supporting logic is to use a fast, microcoded 8-bit CPU to handle this access and the computation. Although there is some speed penalty over separate logic, savings may be realized through a reduced chip count and development time.

7.2.1 C language algorithm.

```
#define MODX 4102


  encodecc( mess,len,k )
  unsigned char mess[] ;     /* the TPDU to be checksummed */
  int      len,
           k;                /* position of first checksum octet
                                as an offset from mess[0]  */
```

```
{ int ip,
      iq,
      ir,
      c0,
      c1;
  unsigned char *p,*p1,*p2,*p3 ;

  p = mess ; p3 = mess + len ;

  if ( k > 0) { mess[k-1] = 0x00 ; mess[k] = 0x00 ; }
       /* insert zeros for checksum octets */

  c0 = 0 ; c1 = 0  ; p1 = mess ;
  while (p1 < p3)    /* outer sum accumulation loop */
  {
   p2 = p1 + MODX ; if (p2 > p3) p2 = p3 ;
   for (p = p1 ; p < p2 ; p++) /*  inner sum accumulation loop */
   { c0 = c0 + (*p) ; c1 = c1 + c0 ;
   }
   c0 = c0%255 ; c1 = c1%255 ; p1 = p2 ;
       /* adjust accumulated sums to mod 255 */
   }
   ip = (c1 << 8) + c0 ;      /* concatenate c1 and c0 */

   if (k > 0)
   {     /* compute and insert checksum octets */
      iq = ((len-k)*c0 - c1)%255 ; if (iq <= 0) iq = iq + 255 ;
      mess[k-1] = iq ;
      ir = (510 - c0 - iq) ;
      if (ir > 255) ir = ir - 255 ; mess[k] = ir ;
    }
    return(ip) ;
  }
```

.2.2 C/assembler algorithm.

```
#include <math>

  encodecm(mess,len,k)
  unsigned char *mess ;
  int      len,k      ;
  {
    int i,ip,c0,c1 ;

    if (k > 0) { mess[k-1] = 0x00 ; mess[k] = 0x00 ; }
    ip = optm1(mess,len,&c0,&c1) ;
    if (k > 0)
    { i = ( (len-k)*c0 - c1)%255 ; if (i <= 0) i = i + 255 ;
      mess[k-1] = i ;
      i = (510 - c0 - i) ; if (i > 255) i = i - 255 ;
```

```
      mess[k] = i ;
    }
    return(ip) ;
  }
;       calling sequence optm(message,length,&c0,&c1) where
;       message is an array of bytes
;       length   is the length of the array
;       &c0 and &c1 are the addresses of the counters to hold the
;       remainder of; the first and second order partial sums
;       mod(255).

        .ENTRY   optm1,^M<r2,r3,r4,r5,r6,r7,r8,r9,r10,r11>
        movl     4(ap),r8       ; r8---> message
        movl     8(ap),r9       ; r9=length
        clrq     r4             ; r5=r4=0
        clrq     r6             ; r7=r6=0
        clrl     r3             ; clear high order bytes of r3
        movl     #255,r10       ; r10 holds the value 255
        movl     #4102,r11      ; r11= MODX
xloop:  movl     r11,r7         ; if r7=MODX
        cmpl     r9,r7          ; is r9>=r7 ?
        bgeq     yloop          ; if yes, go and execute the inner
                                ; loop MODX times.
        movl     r9,r7          ; otherwise set r7, the inner loop
                                ; counter,
yloop:  movb     (r8)+,r3       ;
        addl2    r3,r4          ; suml=suml+byte
        addl2    r4,r6          ; sum2=sum2+suml
        sobgtr   r7,yloop       ; while r7>0 return to iloop
                                ; for mod 255 addition
    ediv     r10,r6,r0,r6   ; r6=remainder
    ediv     r10,r4,r0,r4   ;
    subl2    r11,r9         ; adjust r9
    bgtr     xloop          ; go for another loop if necessary
    movl     r4,@12(ap)     ; first argument
    movl     r6,@16(ap)     ; second argument
    ashl     #8,r6,r0       ;
    addl2    r4,r0          ;
    ret
```

7.2.3 Assembler algorithm.

```
buff0:  .blkb   3               ; allocate 3 bytes so that aloop is
                        ; optimally aligned
;       macro implementation of Fletcher's algorithm.
;       calling sequence ip=encodemm(message,length,k) where
;       message is an array of bytes
;       length   is the length of the array
;       k        is the location of the check octets if >0,
;                an indication not to encode if 0.
;
```

```
movl    4(ap),r8        ; r8---> message
movl    8(ap),r9        ; r9=length
clrq    r4              ; r5=r4=0
clrq    r6              ; r7=r6=0
clrl    r3              ; clear high order bytes of r3
movl    #255,r10        ; r10 holds the value 255
movl    12(ap),r2       ; r2=k
bleq    bloop           ; if r2<=0, we do not encode
subl3   r2,r9,r11       ; set r11=L-k
addl2   r8,r2           ; r2---> octet k+1
clrb    (r2)            ; clear check octet k+1
clrb    -(r2)           ; clear check octet k, r2---> octet k.
bloop:  movw    #4102,r7    ; set r7 (inner loop counter) = to MODX
cmpl    r9,r7           ; if r9>=MODX, then go directly to adjust r9
bgeq    aloop           ; and execute the inner loop MODX times.
movl    r9,r7           ; otherwise set r7, the inner loop counter,
                        ; equal to r9, the number of the
                        ; unprocessed characters
aloop:  movb    (r8)+,r3        ;
addl2   r3,r4           ; c0=c0+byte
addl2   r4,r6           ; sum2=sum2+sum1
sobgtr  r7,aloop        ; while r7>0 return to iloop
                                ; for mod 255 addition
ediv    r10,r6,r0,r6    ; r6=remainder
ediv    r10,r4,r0,r4    ;
subl2   #4102,r9        ;
bgtr    bloop           ; go for another loop if necessary
ashl    #8,r6,r0        ; r0=256*r6
addl2   r4,r0           ; r0=256*r6+r4
cmpl    r2,r7           ; since r7=0, we are checking if r2 is
bleq    exit            ; zero or less: if yes we bypass
                                ; the encoding.
movl    r6,r8           ; r8=c1
mull3   r11,r4,r6       ; r6=(L-k)*c0
ediv    r10,r6,r7,r6    ; r6 = (L-k)*c0 mod(255)
subl2   r8,r6           ; r6= ((L-k)*c0)%255 -c1 and if negative,
bgtr    byte1           ; we must
addl2   r10,r6          ; add 255
byte1:  movb    r6,(r2)+ ; save the octet and let r2---> octet k+1
addl2   r6,r4           ; r4=r4+r6=(x+c0)
subl3   r4,r10,r4       ; r4=255-(x+c0)
bgtr    byte2           ; if >0 r4=octet (k+1)
addl2   r10,r4          ; r4=255+r4
byte2:  movb    r4,(r2)         ; save y in octet k+1
exit:   ret
```

3 Parameter selection.

3.1 Connection control.

Expressions for timer values used to control the general transport

connection behavior are given in IS 8073. However, values for the specific factors in the expressions are not given and the expressions are only estimates. The derivation of timer values from these expressions is not mandatory in the standard. The timer value expressions in IS 8073 are for a connection-oriented network service and may not apply to a connectionless network service.

The following symbols are used to denote factors contributing to timer values, throughout the remainder of this Part.

Elr = expected maximum transit delay, local to remote

Erl = expected maximum transit delay, remote to local

Ar = time needed by remote entity to generate an acknowledgement

Al = time needed by local entity to generate an acknowledgement

x = local processing time for an incoming TPDU

Mlr = maximum NSDU lifetime, local to remote

Mrl = maximum NSDU lifetime, remote to local

Tl = bound for maximum time local entity will wait for acknowledgement before retransmitting a TPDU

R = bound for maximum local entity will continue to transmit a TPDU that requires acknowledgment

N = bound for maximum number of times local entity will transmit a TPDU requiring acknowledgement

L = bound for the maximum time between the transmission of a TPDU and the receipt of any acknowledgment relating to it.

I = bound for the time after which an entity will initiate procedures to terminate a transport connection if a TPDU is not received from the peer entity

W = bound for the maximum time an entity will wait before transmitting up-to-date window information

These symbols and their definitions correspond to those given in Clause 12 of IS 8073.

8.1.1 Give-up timer.

The give-up timer determines the amount of time the transport entity will continue to await an acknowledgement (or other appropriate reply) of a transmitted message after the message

has been retransmitted the maximum number of times. The recommendation given in IS 8073 for values of this timer is expressed by

T1 + W + Mrl, for DT and ED TPDUs

T1 + Mrl, for CR, CC, and DR TPDUs,

where

T1 = Elr + Erl + Ar + x.

However, it should be noted that Ar will not be known for either the CR or the CC TPDU, and that Elr and Erl may vary considerably due to routing in some conectionless network services. In Part 8.3.1, the determination of values for T1 is discussed in more detail. Values for Mrl generally are relatively fixed for a given network service. Since Mrl is usually much larger than expected values of T1, a rule-of-thumb for the give-up timer value is 2*Mrl + Al + x for the CR, CC and DR TPDUs and 2*Mrl + W for DT and ED TPDUs.

8.1.2 Inactivity timer.

This timer measures the maximum time period during which a transport connection can be inactive, i.e., the maximum time an entity can wait without receiving incoming messages. A usable value for the inactivity timer is

I = 2*(max(T1,W)*N).

This accounts for the possibility that the remote peer is using a window timer value different from that of the local peer. Note that an inactivity timer is important for operation over connectionless network services, since the periodic receipt of AK TPDUs is the only way that the local entity can be certain that its peer is still functioning.

8.1.3 Window timer.

The window timer has two purposes. It is used to assure that the remote peer entity periodically receives the current state of the local entity's flow control, and it ensures that the remote peer entity is aware that the local entity is still functioning. The first purpose is necessary to place an upper bound on the time necessary to resynchronize the flow control should an AK TPDU which notifies the remote peer of increases in credit be lost. The second purpose is necessary to prevent the inactivity timer of the remote peerfrom expiring. The value for the window timer, W, depends on several factors, among which are the transit delay, the acknowledgement strategy, and the probability of TPDU loss in the network. Generally, W should satisfy the following condition:

$$W > C*(Erl + x)$$

where C is the maximum amount of credit offered. The rationale for this condition is that the right-hand side represents the maximum time for receiving the entire window. The protocol requires that all data received be acknowledged when the upper edge of the window is seen as a sequence number in a received DT TPDU. Since the window timer is reset each time an AK TPDU is transmitted, there is usually no need to set the timer to any less than the value on the right-hand side of the condition. An exception is when both C and the maximum TPDU size are large, and Erl is large.

When the probability that a TPDU will be lost is small, the value of W can be quite large, on the order of several minutes. However, this increases the delay the peer entity will experience in detecting the deactivation of the local transport entity. Thus, the value of W should be given some consideration in terms of how soon the peer entity needs to detect inactivity. This could be done by placing such information into a quality of service record associated with the peer's address.

When the expected network error rate is high, it may be necessary to reduce the value of W to ensure that AK TPDUs are being received by the remote entity, especially when both entities are quiescent for some period of time.

8.1.4 Reference timer.

The reference timer measures the time period during which a source reference must not be reassigned to another transport connection, in order that spurious duplicate messages not interfere with a new connection. The value for this timer given in IS 8073 is

$$L = Mlr + Mrl + R + Ar$$

where

$$R = Tl*N + z$$

in which z is a small tolerance quantity to allow for factors internal to the entity. The use of L as a bound, however, must be considered carefully. In some cases, L may be very large, and not realistic as an upper or a lower bound. Such cases may be encountered on routes over several catenated networks where R is set high to provide adequate recovery from TPDU loss. In other cases L may be very small, as when transmission is carried out over a LAN and R is set small due to low probability of TPDU loss. When L is computed to be very small, the reference need not be timed out at all, since the probability of interference is zero. On the other hand, if L is computed to be very large a smaller value can be used.

One choice for the value might be

L = min(R,(Mrl + Mlr)/2)

If the reference number assigned to a new connection by an entity is monotonically incremented for each new connection through the entire available reference space (maximum 2**16 - 1), the timer is not critical: the sequence space is large enough that it is likely that there will be no spurious messages in the network by the time reference numbers are reused.

8.2 Flow control.

The peer-to-peer flow control mechanism in the transport protocol determines the upper bound on the pace of data exchange that occurs on transport connections. The transport entity at each end of a connection offers a credit to its peer representing the number of data messages it is currently willing to accept. All received data messages are acknowledged, with the acknowledgement message containing the current receive credit information. The three credit allocation schemes discussed below present a diverse set of examples of how one might derive receive credit values.

8.2.1 Pessimistic credit allocation.

Pessimistic credit allocation is perhaps the simplest form of flow control. It is similar in concept to X-on/X-off control. In this method, the receiver always offers a credit of one TPDU. When the DT TPDU is received, the receiver responds with an AK TPDU carrying a credit of zero. When the DT TPDU has been processed by the receiving entity, an additional AK TPDU carrying a credit of one will be sent. The advantage to this approach is that the data exchange is very tightly controlled by the receiving entity. The disadvantages are: 1) the exchange is slow, every data message requiring at least the time of two round trips to complete the transfer transfer, and 2) the ratio of acknowledgement to data messages sent is 2:1. While not recommeneded, this scheme illustrates one extreme method of credit allocation.

8.2.2 Optimistic credit allocation.

At the other extreme from pessimistic credit allocation is optimistic credit allocation, in which the receiver offers more credit than it has buffers. This scheme has two dangers. First, if the receiving user is not accepting data at a fast enough rate, the receiving transport's buffers will become filled. Since the credit offered was optimistic, the sending entity will continue to transmit data, which must be dropped by the receiving entity for lack of buffers. Eventually, the sender may reach the maximum number of retransmissions and terminate the connection.

The second danger in using optimistic flow control is that the sending entity may transmit faster than the receiving entity can consume. This could result from the sender being implemented on a faster machine or being a more efficient implementation. The resultant behavior is essentially the same as described above: receive buffer saturation, dropped data messages, and connection termination.

The two dangers cited above can be ameliorated by implementing the credit reduction scheme as specified in the protocol. However, optimistic credit allocation works well only in limited circumstances. In most situations it is inappropriate and inefficient even when using credit reduction. Rather than seeking to avoid congestion, optimistic allocation causes it, in most cases, and credit reduction simply allows one to recover from congestion once it has happened. Note that optimistic credit allocation combined with caching out-of-sequence messages requires a sophisticated buffer management scheme to avoid reasssembly deadlock and subsequent loss of the transport connection.

8.2.3 Buffer-based credit allocation.

Basing the receive credit offered on the actual availability of receive buffers is a better method for achieving flow control. Indeed, with few exceptions, the implementations that have been studied used this method. It continuous flow of data and eliminating the need for the credit-restoring acknowledgements. Since only available buffer space is offered, the dangers of optimistic credit allocation are also avoided.

The amount of buffer space needed to maintain a continuous bulk data transfer, which represents the maximum buffer requirement, is dependent on round trip delay and network speed. Generally, one would want the buffer space, and hence the credit, large enough to allow the sender to send continuously, so that incremental credit updates arrive just prior to the sending entity exhausting the available credit. One example is a single-hop satellite link operating at 1.544 Mbits/sec. One report [COL85] indicates that the buffer requirement necessary for continuous flow is approximately 120 Kbytes. For 10 Mbits/sec. IEEE 802.3 and 802.4 LANs, the figure is on the order of 10K to 15K bytes [BRI85, INT85, MIL85].

An interesting modification to the buffer-based credit allocation scheme is suggested by R.K. Jain [JAI85]. Whereas the approach described above is based strictly on the available buffer space, Jain suggests a scheme in which credit is reduced voluntarily by the sending entity when network congestion is detected. Congestion is implied by the occurrence of retransmissions. The sending entity, recognizing retransmissions, reduces the local value of credit to one, slowly raising it to the actual receive credit allocation as error-free transmissions continue to occur. This

technique can overcome various types of network congestion occurring when a fast sender overruns a slow receiver when no link level flow control is available.

8.2.4 Acknowledgement policies.

It is useful first to review the four uses of the acknowledgement message in Class 4 transport. An acknowledgement message:

1) confirms correct receipt of data messages,

2) contains a credit allocation, indicating how many data messages the entity is willing to receive from the correspondent entity,

3) may optionally contain fields which confirm receipt of critical acknowledgement messages, known as flow control confirmation (FCC), and

4) is sent upon expiration of the window timer to maintain a minimum level of traffic on an otherwise quiescent connection.

In choosing an acknowledgement strategy, the first and third uses mentioned above, data confirmation and FCC, are the most relevant; the second, credit allocation, is determined according to the flow control strategy chosen, and the fourth, the window acknowledgement, is only mentioned briefly in the discussion on flow control confirmation.

8.2.4.1 Acknowledgement of data.

The primary purpose of the acknowledgement message is to confirm correct receipt of data messages. There are several choices that the implementor must make when designing a specific implementation. Which choice to make is based largely on the operating environment (e.g., network error characteristics). The issues to be decided upon are discussed in the sections below.

8.2.4.1.1 Misordered data messages.

Data messages received out of order due to network misordering or loss can be cached or discarded. There is no single determinant that guides the implementor to one or the other choice. Rather, there are a number of issues to be considered.

One issue is the importance of maintaining a low delay as perceived by the user. If transport data messages are lost or damaged in transit, the absence of a positive acknowledgement will trigger a retransmission at the sending entity. When the retransmitted data message arrives at the receiving transport, it can be delivered

to the user. If subsequent data messages had been cached, they could be delivered to the user at the same time. The delay between the sending and receiving users would, on average, be shorter than if messages subsequent to a lost message were dependent on retransmission for recovery.

A second factor that influences the caching choice is the cost of transmission. If transmission costs are high, it is more economical to cache misordered data, in conjunction with the use of selective acknowledgement (described below), to avoid retransmissions.

There are two resources that are conserved by not caching misordered data: design and implementation time for the transport entity and CPU processing time during execution. Savings in both categories accrue because a non-caching implementation is simpler in its buffer management. Data TPDUs are discarded rather than being reordered. This avoids the overhead of managing the gaps in the received data sequence space, searching of sequenced message lists, and inserting retransmitted data messages into the lists.

8.2.4.1.2 Nth acknowledgement.

In general, an acknowledgement message is sent after receipt of every N data messages on a connection. If N is small compared to the credit offered, then a finer granularity of buffer control is afforded to the data sender's buffer management function. Data messages are confirmed in small groups, allowing buffers to be reused sooner than if N were larger. The cost of having N small is twofold. First, more acknowledgement messages must be generated by one transport entity and processed by another, consuming some of the CPU resource at both ends of a connection. Second, the acknowledgement messages consume transmission bandwidth, which may be expensive or limited.

For larger N, buffer management is less efficient because the granularity with which buffers are controlled is N times the maximum TPDU size. For example, when data messages are transmitted to a receiving entity employing this strategy with large N, N data messages must be sent before an acknowledgement is returned (although the window timer causes the acknowledgement to be sent eventually regardless of N). If the minimum credit allocation for continuous operation is actually a fraction of N, a credit of N must still be offered, and N receive buffers reserved, to achieve a continuous flow of data messages. Thus, more receive buffers are used than are actually needed. (Alternatively, if one relies on the timer, which must be adjusted to the receipt time for N and will not expire until some time after the fraction of N has been sent, there may be idle time.)

The choice of values for N depends on several factors. First, if the

rate at which DT TDPUs are arriving is relatively low, then there is not much justification for using a value for N that exceeds 2. On the other hand, if the DT TPDU arrival rates is high or the TPDU's arrive in large groups (e.g., in a frame from a satellite link), then it may be reasonable to use a larger value for N, simply to avoid the overhead of generating and sending the acknowledgements while procesing the DT TPDUs. Second, the value of N should be related to the maximum credit to be offered. Letting C be the maximum credit to be offered, one should choose N < C/2, since the receipt of C TPDUs without acknowledging will provoke sending one in any case. However, since the extended formats option for transport provides max C = 2**16 - 1, a choice of N = 2**15 - 2 is likely to cause some of the sender's retransmission timers to expire. Since the retransmitted TPDU's will arrive out of sequence, they will provoke the sending of AK TPDU's. Thus, not much is gained by using an N large. A better choice is N = log C (base 2). Third, the value of should be related to the maximum TPDU size used on the connection and the overall buffer management. For example, the buffer management may be tied to the largest TPDU that any connection will use, with each connection managing the actual way in which the negotiated TPDU size relates to this buffer size. In such case, if a connection has negotiated a maximum TPDU size of 128 octets and the buffers are 2048 octets, it may provide better management to partially fill a buffer before acknowledging. If the example connection has two buffers and has based offered credit on this, then one choice for N could be 2*log(2048/128) = 8. This would mean that an AK TPDU would be sent when a buffer is half filled (2048/128 = 16), and a double buffering scheme used to manage the use of the two buffers. the use of the t There are two studies which indicate that, in many cases, 2 is a good choice for N [COL85, BRI85]. The increased granularity in buffer management is reasonably small when compared to the credit allocation, which ranges from 8K to 120K octets in the studies cited. The benefit is that the number of acknowledgements generated (and consumed) is cut approximately in half.

8.2.4.1.3 Selective acknowledgement.

Selective acknowledgement is an option that allows misordered data messages to be confirmed even in the presence of gaps in the received message sequence. (Note that selective acknowledgement is only meaningul whe caching out-of-orderdata messags.) The advantage to using this mechanism is hat i grealy reduces the number of unnecessary retransmissions, thus saving both computing time and transmission bandwidth [COL85] (see the discussion in Part 8.2.4.1.1 for more details).

8.2.4.2 Flow control confirmation and fast retransmission.

Flow control confirmation (FCC) is a mechanism of the transport protocol whereby acknowledgement messages containing critical flow control information are confirmed. The critical acknowledgement

messages are those that open a closed flow control window and certain ones that occur subsequent to a credit reduction. In principle, if these critical messages are lost, proper resynchroniztion of the flow control relies on the window timer, which is generally of relatively long duration. In order to reduce delay in resynchronizing the flow control, the receiving entity can repeatedly send, within short intervals, AK TPDUs carrying a request for confirmation of the flow control state, a procedure known as "fast" retransmission (of the acknowledgement). If the sender responds with an AK TPDU carrying an FCC parameter, fast retransmission is halted. If no AK TPDU carrying the FCC parameter is received, the fast transmission halts after having reached a maximum number of retransmissions, and the window timer resumes control of AK TPDU transmission. It should be noted that FCC is an optional mechanism of transport and the data sender is not required to respond to a request for confirmation of the flow control state wih an AK TPDU carrying the FCC parameter.

Some considerations for deciding whether or not to use FCC and fast retransmisson procedures are as follows:

1) likelihood of credit reduction on a given transport connection;

2) probability of TPDU loss;

3) expected window timer period;

4) window size; and

5) acknowledgement strategy.

At this time, there is no reported experience with using FCC and fast retransmission. Thus, it is not known whether or not the procedures produce sufficient reduction of resynchronization delay to warrant implementing them.

When implementing fast retransmission, it is suggested that the timer used for the window timer be employed as the fast timer, since the window is disabled during fast retransmission in any case. This will avoid having to manage another timer. The formal description expressed the fast retransmission timer as a separate timer for clarity.

8.2.4.3 Concatenation of acknowledgement and data.

When full duplex communication is being operated by two transport entities, data and acknowledgement TPDUs from each one of the entities travel in the same direction. The transport protocol permits concatenating AK TPDUs in the same NSDU as a DT TPDU. The advantage of using this feaure in an implementation is that fewer NSDUs will be transmitted, and, consequently, fewer total octets will

be sent, due to the reduced number of network headers transmitted. However, when operating over the IP, this advantage may not necessarily be recognized, due to the possible fragmentation of the NSDU by the IP. A careful analysis of the treatment of the NSDU in internetwork environments should be done to determine whether or not concatenation of TPDUs is of sufficient benefit to justify its use in that situation.

8.2.5 Retransmission policies.

There are primarily two retransmission policies that can be employed in a transport implementation. In the first of these, a separate retransmission timer is initiated for each data message sent by the transport entity. At first glance, this approach appears to be simple and straightforward to implement. The deficiency of this scheme is that it is inefficient. This derives from two sources. First, for each data message transmitted, a timer must be initiated and cancelled, which consumes a significant amount of CPU processing time [BRI85]. Second, as the list of outstanding timers grows, management of the list also becomes increasingly expensive. There are techniques which make list management more efficient, such as a list per connection and hashing, but implementing a policy of one retransmission timer per transport connection is a superior choice.

The second retransmission policy, implementing one retransmission timer for each transport conenction, avoids some of the inefficiencies cited above: the list of outstanding timers is shorter by approximately an order of magnitude. However, if the entity receiving the data is generating an acknowledgement for every data message, the timer must still be cancelled and restarted for each data/acknowledgement message pair (this is an additional impetus for implementing an Nth acknowledgement policy with N=2).

The rules governing the single timer per connection scheme are listed below.

1) If a data message is transmitted and the retransmission timer for the connection is not already running, the timer is started.

2) If an acknowledgement for previously unacknowledged data is received, the retransmission timer is restarted.

3) If an acknowledgement message is received for the last outstanding data message on the connection then the timer is cancelled.

4) If the retransmission timer expires, one or more unacknowledged data messages are retransmitted, beginning with the one sent earliest. (Two

reports [HEA85, BRI85] suggest that the number to retransmit is one.)

8.3 Protocol control.

8.3.1 Retransmission timer values.

8.3.1.1 Data retransmission timer.

The value for the reference timer may have a significant impact on the performance of the transport protocol [COL85]. However, determining the proper value to use is sometimes difficult. According to IS 8073, the value for the timer is computed using the transit delays, Erl and Elr, the acknowledgement delay, Ar, and the local TPDU processing time, x:

T1 = Erl + Elr + Ar + x

The difficulty in arriving at a good retransmission timer value is directly related to the variability of these factors Of the two, Erl and Elr are the most susceptible to variation, and therefore have the most impact on determining a good timer value. The following paragraphs discuss methods for choosing retransmission timer values that are appropriate in several network environments.

In a single-hop satellite environment, network delay (Erl or Elr) has small variance because of the constant propagation delay of about 270 ms., which overshadows the other components of network delay. Consequently, a fixed retransmission timer provides good performance. For example, for a 64K bit/sec. link speed and network queue size of four, 650 ms. provides good performance [COL85].

Local area networks also have constant propagation delay. However, propagation delay is a relatively unimportant factor in total network delay for a local area network. Medium access delay and queuing delay are the significant components of network delay, and (Ar + x) also plays a significant role in determining an appropriate retransmission timer. From the discussion presented in Part 3.4.3.2 typical numbers for (Ar + x) are on the order of 5 - 6.5 ms and for Erl or Elr, 5 - 35 ms. Consequently, a reasonable value for the retransmission timer is 100 ms. This value works well for local area networks, according to one cited report [INT85] and simulation work performed at the NBS.

For better performance in an environment with long propagation delays and significant variance, such as an internetwork an adaptive algorithm is preferred, such as the one suggested value for TCP/IP [ISI81]. As analyzed by Jain [JAI85], the algorithm uses an exponential averaging scheme to derive a round trip delay estimate:

$$D(i) = b * D(i-1) + (1-b) * S(i)$$

where D(i) is the update of the delay estimate, S(i) is the sample round trip time measured between transmission of a given packet and receipt of its acknowledgement, and b is a weighting factor between 0 and 1, usually 0.5. The retransmission timer is expressed as some multiplier, k, of D. Small values of k cause quick detection of lost packets, but result in a higher number of false timeouts and, therefore, unnecessary retransmissions. In addition, the retransmission timer should be increased arbitrarily for each case of multiple transmissions; an exponential increase is suggested, such that

$$D(i) = c * D(i-1)$$

where c is a dimensionless parameter greater than one.

The remaining parameter for the adaptive algorithm is the initial delay estimate, D(0). It is preferable to choose a slightly larger value than needed, so that unnecessary retransmissions do not occur at the beginning. One possibility is to measure the round trip delay during connection establishment. In any case, the timer converges except under conditions of sustained congestion.

8.3.1.2 Expedited data retransmission timer.

The timer which governs retransmission of expedited data should be set using the normal data retransmission timer value.

8.3.1.3 Connect-request/confirm retransmission timer.

Connect request and confirm messages are subject to Erl + Elr, total network delay, plus processing time at the receiving transport entity, if these values are known. If an accurate estimate of the round trip time is not known, two views can be espoused in choosing the value for this timer. First, since this timer governs connection establishment, it is desirable to minimize delay and so a small value can be chosen, possibly resulting in unnecessary retransmissions. Alternatively, a larger value can be used, reducing the possibility of unnecessary retransmissions, but resulting in longer delay in connection establishment should the connect request or confirm message be lost. The choice between these two views is dictated largely by local requirements.

8.3.1.4 Disconnect-request retransmission timer.

The timer which governs retransmission of the disconnect request message should be set from the normal data retransmission timer value.

8.3.1.5 Fast retransmission timer.

The fast retransmission timer causes critical acknowledgement

messages to be retransmitted avoiding delay in resynchronizing credit. This timer should be set to approximately Erl + Elr.

8.3.2 Maximum number of retransmissions.

This transport parameter determines the maximum number of times a data message will be retransmitted. A typical value is eight. If monitoring of network service is performed then this value can be adjusted according to observed error rates. As a high error rate implies a high probability of TPDU loss, when it is desirable to continue sending despite the decline in quality of service, the number of TPDU retransmissions (N) should be increased and the retransmission interval (T1) reduced.

8.4 Selection of maximum Transport Protocol data unit size.

The choice of maximum size for TPDUs in negotiation proposals depends on the application to be served and the service quality of the supporting network. In general, an application which produces large TSDUs should use as large TPDUs as can be negotiated, to reduce the overhead due to a large number of small TPDUs. An application which produces small TSDUs should not be affected by the choice of a large maximum TPDU size, since a TPDU need not be filled to the maximum size to be sent. Consequently, applications such as file transfers would need larger TPDUs while terminals would not. On a high bandwidth network service, large TPDUs give better channel utilization than do smaller ones. However, when error rates are high, the likelihood for a given TPDU to be damaged is correlated to the size and the frequency of the TPDUs. Thus, smaller TPDU size in the condition of high error rates will yield a smaller probability that any particular TPDU will be lost.

The implementor must choose whether or not to apply a uniform maximum TPDU size to all connections. If the network service is uniform in service quality, then the selection of a uniform maximum can simplify the implementation. However, if the network quality is not uniform and it is desirable to optimize the service provided to the transport user as much as possible, then it may be better to determine the maximum size on an individual connection basis. This can be done at the time of the network service access if the characteristics of the subnetwork are known.

NOTE: The maximum TPDU size is important in the calculation of the flow control credit, which is in numbers of TPDUs offered. If buffer space is granted on an octet base, then credit must be granted as buffer space divided by maximum TPDU size. Use of a smaller TPDU size can be equivalent to optimistic credit allocation and can lead to the expected problems, if proper analysis of the management is not done.

9 Special options.

Special options may be obtained by taking advantage of the manner in which IS 8073 and N3756 have been written. It must be emphasized that these options in no way violate the intentions of the standards bodies that produced the standards. Flexibility was deliberately written into the standards to ensure that they do not constrain applicability to a wide variety of situations.

9.1 Negotiations.

The negotiation procedures in IS 8073 have deliberate ambiguities in them to permit flexibility of usage within closed groups of communicants (the standard defines explicitly only the behavior among open communicants). A closed group of communicants in an open system is one which, by reason of organization, security or other special needs, carries on certain communication among its members which is not of interest or not accessible to other open system members. Examples of some closed groups within DOD might be: an Air Force Command, such as the SAC; a Navy base or an Army post; a ship; Defense Intelligence; Joint Chiefs of Staff. Use of this characteristic does not constitute standard behavior, but it does not violate conformance to the standard, since the effects of such usage are not visible to non-members of the closed group. Using the procedures in this way permits options not provided by the standard. Such options might permit, for example, carrying special protection codes on protocol data units or for identifying DT TPDUs as carrying a particular kind of message.

Standard negotiation procedures state that any parameter in a received CR TPDU that is not defined by the standard shall be ignored. This defines only the behavior that is to be exhibited between two open systems. It does not say that an implementation which recognizes such non-standard parameters shall not be operated in networks supporting open systems interconnection. Further, any other type TPDU containing non-standard parameters is to be treated as a protocol error when received. The presumption here is that the non-standard parameter is not recognized, since it has not been defined. Now consider the following example:

Entity A sends Entity B a CR TPDU containing a non-standard parameter.

Entity B has been implemented to recognize the non-standard parameter and to interpret its presence to mean that Entity A will be sending DT TPDUs to Entity B with a special protection identifier parameter included.

Entity B sends a CC TPDU containing the non-standard parameter to indicate to Entity A that it has received and understood the parameter, and is prepared to receive the specially marked DT TPDUs

from Entity A. Since Entity A originally sent the non-standard parameter, it recognizes the parameter in the CC TPDU and does not treat it as a protocol error.

Entity A may now send the specially marked DT TPDUs to Entity B and Entity B will not reject them as protocol errors.

Note that Entity B sends a CC TPDU with the non-standard parameter only if it receives a CR TPDU containing the parameter, so that it does not create a protocol error for an initiating entity that does not use the parameter. Note also that if Entity B had not recognized the parameter in the CR TPDU, it would have ignored it and not returned a CC TPDU containing the parameter. This non-standard behavior is clearly invisible and inaccessible to Transport entities outside the closed group that has chosen to implement it, since they are incapable of distinguishing it from errors in protocol.

9.2 Recovery from peer deactivation.

Transport does not directly support the recovery of the transport connection from a crashed remote transport entity. A partial recovery is possible, given proper interpretation of the state tables in Annex A to IS 8073 and implementation design. The interpretation of the Class 4 state tables necessary to effect this operation is as follows:

Whenever a CR TPDU is received in the state OPEN, the entity is required only to record the new network connection and to reset the inactivity timer. Thus, if the initiator of the original connection is the peer which crashed, it may send a new CR TPDU to the surviving peer, somehow communicating to it the original reference numbers (there are several ways that this can be done).

Whenever a CC TPDU is received in the

state OPEN, the receiver is required only to record the new network connection, reset the inactivity timer and send either an AK, DT or ED TPDU. Thus, if the responder for the original connection is the peer which crashed, it may send a new CC TPDU to the surviving peer, communicating to it the original reference numbers.

In order for this procedure to operate properly, the situation in a., above, requires a CC TPDU to be sent in response. This could be the original CC TPDU that was sent, except for new reference numbers. The original initiator will have sent a new reference number in the new CR TPDU, so this would go directly into the CC TPDU to be returned. The new reference number for the responder could just be a new assignment, with the old reference number frozen. In the situation in b., the originator could retain its reference number (or

assign a new one if necessary), since the CC TPDU should carry both old reference numbers and a new one for the responder (see below). In either situation, only the new reference numbers need be extracted from the CR/CC TPDUs, since the options and parameters will have been previously negotiated. This procedure evidently requires that the CR and CC TPDUs of each connection be stored by the peers in nonvolatile memory, plus particulars of the negotiations.

To transfer the new reference numbers, it is suggested that the a new parameter in the CR and CC TPDU be defined, as in Part 9.1, above. This parameter could also carry the state of data transfer, to aid in resynchronizing, in the following form:

1) the last DT sequence number received by the peer that crashed;

2) the last DT sequence number sent by the peer that crashed;

3) the credit last extended by the peer that crashed;

4) the last credit perceived as offered by the surviving peer;

5) the next DT sequence number the peer that crashed expects to send (this may not be the same as the last one sent, if the last one sent was never acknowledged);

6) the sequence number of an unacknowledged ED TPDU, if any;

7) the normal data sequence number corresponding to the transmission of an unacknowledged ED TPDU, if any (this is to ensure the proper ordering of the ED TPDU in the normal data flow);

A number of other considerations must be taken into account when attempting data transfer resynchronization. First, the recovery will be greatly complicated if subsequencing or flow control confirmation is in effect when the crash occurs. Careful analysis should be done to determine whether or not these features provide sufficient benefit to warrant their inclusion in a survivable system. Second, non-volatile storage of TPDUs which are unacknowledged must be used in order that data loss at the time of recovery can be minimized. Third, the values for the retranmsission timers for the communicating peers must allow sufficient time for the recovery to be attempted. This may result in longer delays in retransmitting when TPDUs are lost under normal conditions. One way that this might be achieved is for the peers to exchange in the original CR/CC TPDU exchange, their expected lower bounds for the retransmission timers, following the procedure in Part 9.1. In this manner, the peer that crashed may be determine whether or not a new connection should be attempted. Fourth, while the recovery involves directly only the transport peers when operating over a connectionless network service, recovery when

operating over a connection-oriented network service requires some sort of agreement as to when a new network connection is to be established (if necessary) and which peer is responsible for doing it. This is required to ensure that unnecessary network connections are not opened as a result of the recovery. Splitting network connections may help to ameliorate this problem.

9.3 Selection of transport connection reference numbers.

In N3756, when the reference wait period for a connection begins, the resources associated with the connection are released and the reference number is placed in a set of frozen references. A timer associated with this number is started, and when it expires, the number is removed from the set. A function which chooses reference numbers checks this set before assigning the next reference number. If it is desired to provide a much longer period by the use of a large reference number space, this can be met by replacing the implementation dependent function "select_local_ref" (page TPE-17 of N3756) by the following code:

```
function select_local_ref : reference_type;

begin
last_ref := (last_ref + 1) mod( N+1 ) + 1;
while last_ref in frozen_ref[class_4] do
          last_ref := (last_ref + 1) mod( N+1 ) + 1;
select_local_ref := last_ref;
end;
```

where "last_ref" is a new variable to be defined in declarations (pages TPE-10 - TPE-11), used to keep track of the last reference value assigned, and N is the length of the reference number cycle, which cannot exceed 2**16 - 1 since the reference number fields in TPDUs are restricted to 16 bits in length.

9.4 Obtaining Class 2 operation from a Class 4 implementation.

The operation of Class 4 as described in IS 8073 logically contains that of the Class 2 protocol. The formal description, however, is written assuming Class 4 and Class 2 to be distinct. This was done because the description must reflect the conformance statement of IS 8073, which provides that Class 2 alone may be implemented.

However, Class 2 operation can be obtained from a Class 4 implementation, which would yield the advantages of lower complexity, smaller memory requirements, and lower implementation costs as compared to implementing the classes separately. The implementor will have to make the following provisions in the transport entity and the Class 4 transport machine to realize Class 2 operation.

1) Disable all timers. In the formal description, all Class 4 timers except the reference timer are in the Class 4 TPM. These timers can be designed at the outset to be enabled or not at the instantiation of the TPM. The reference timer is in the Transport Entity module (TPE) and is activated by the TPE recognizing that the TPM has set its "please_kill_me" variable to "freeze". If the TPM sets this variable instead to "now", the reference timer for that transport connection is never started. However, IS 8073 provides that the reference timer can be used, as a local entity management decision, for Class 2.

   The above procedure should be used when negotiating from Class 4 to Class 2. If Class 2 is proposed as the preferred class, then it is advisable to not disable the inactivity timer, to avoid the possibility of deadlock during connection establishment if the peer entity never responds to the CR TPDU. The inactivity timer should be set when the CR TPDU is sent and deactivated when the CC TPDU is received.

2) Disable checksums. This can be done simply by ensuring that the boolean variable "use_checksums" is always set to "false" whenever Class 2 is to be proposed or negotiated.

3) Never permit flow control credit reduction. The formal description makes flow control credit management a function of the TPE operations and such management is not reflected in the operation of the TPM. Thus, this provision may be handled by always making the "credit-granting" mechanism aware of the class of the TPM being served.

4) Include Class 2 reaction to network service events. The Class 4 handling of network service events is more flexible than that of Class 2 to provide the recovery behavior characteristic of Class 4. Thus, an option should be provided on the handling of N_DISCONNECT_indication and N_RESET_indication for Class 2 operation. This consists of sending a T_DISCONNECT_indication to the Transport User, setting "please_kill_me" to "now" (optionally to "freeze"), and transitioning to the CLOSED state, for both events. (The Class 4 action in the case of the N_DISCONNECT is to remove the network connection from the set of those associated with the transport connection and to attempt to obtain a new network connection if the set becomes empty. The action on receipt of the N_RESET is to do nothing, since the TPE has already issued the N_RESET_response.)

5) Ensure that TPDU parameters conform to Class 2. This implies that subsequence numbers should not be used on AK TPDUs, and no flow control confirmation parameters should ever appear in an AK TPDU. The checksum parameter is prevented from

appearing by the "false" value of the "use_checksums" variable. (The acknowledgement time parameter in the CR and CC TPDUs will not be used, by virtue of the negotiation procedure. No special assurance for its non-use is necessary.)

The TPE management of network connections should see to it that splitting is never attempted with Class 4 TPMs running as Class 2. The handling of multiplexing is the same for both classes, but it is not good practice to multiplex Class 4 and Class 2 together on the same network connection.

10 References.

[BRI85] Bricker, A., L. Landweber, T. Lebeck, M. Vernon, "ISO Transport Protocol Experiments," Draft Report prepared by DLS Associates for the Mitre Corporation, October 1985.

[COL85] Colella, Richard, Marnie Wheatley, Kevin Mills, "COMSAT/NBS Experiment Plan for Transport Protocol," NBS, Report No. NBSIR 85-3141, May 1985.

[CHK85] Chernik, C. Michael, "An NBS Host to Front End Protocol," NBSIR 85-3236, August 1985.

[CHO85] Chong, H.Y., "Software Development and Implementation of NBS Class 4 Transport Protocol," October 1985 (available from the author).

[HEA85] Heatley, Sharon, Richard Colella, "Experiment Plan: ISO Transport Over IEEE 802.3 Local Area Network," NBS, Draft Report (available from the authors), October 1985.

[INT85] "Performance Comparison Between 186/51 and 552," The Intel Corporation, Reference No. COM,08, January 1985.

[ISO84a] IS 8073 Information Processing - Open Systems Interconnection - Transport Protocol Specification, available from ISO TC97/SC6 Secretariat, ANSI, 1430 Broadway, New York, NY 10018.

[ISO84b] IS 7498 Information Processing - Open Systems Interconnection - Basic Reference Model, available from ANSI, address above.

[ISO85a] DP 9074 Estelle - A Formal Description Technique Based on an Extended State Transition Model, available from ISO TC97/SC21 Secretariat, ANSI, address above.

[ISO85b] N3756 Information Processing - Open Systems Interconnection - Formal Description of IS 8073 in Estelle. (Working Draft, ISO TC97/SC6)

[ISO85c] N3279 Information Processing - Open Systems Interconnection - DAD1, Draft Addendum to IS 8073 to Provide a Network Connection Management Service, ISO TC97/SC6 N3279, available from SC6 Secretariat, ANSI, address above.

[JAI85] Jain, Rajendra K., "CUTE: A Timeout Based Congestion Control Scheme for Digitial Network Architecture," Digital Equipment Corporation (available from the author), March 1985.

[LIN85] Linn, R.J., "The Features and Facilities of Estelle," Proceedings of the IFIP WG 6.1 Fifth International Workshop on Protocol Specification, Testing and Verification, North Holland Publishing, Amsterdam, June 1985.

[MIL85a] Mills, Kevin L., Marnie Wheatley, Sharon Heatley, "Predicting Transport Protocol Performance", (in preparation).

[MIL85b] Mills, Kevin L., Jeff Gura, C. Michael Chernik, "Performance Measurement of OSI Class 4 Transport Implementations," NBSIR 85-3104, January 1985.

[NAK85] Nakassis, Anastase, "Fletcher's Error Detection Algorithm: How to Implement It Efficiently and How to Avoid the Most Common Pitfalls," NBS, (in preparation).

[NBS83] "Specification of a Transport Protocol for Computer Communications, Volume 3: Class 4 Protocol," February 1983 (available from the National Technical Information Service).

[NTA84] Hvinden, Oyvind, "NBS Class 4 Transport Protocol, UNIX 4.2 BSD Implementation and User Interface Description," Norwegian Telecommunications Administration Establishment, Technical Report No. 84-4053, December 1984.

[NTI82] "User-Oriented Performance Measurements on the ARPANET: The Testing of a Proposed Federal Standard," NTIA Report 82-112 (available from NTIA, Boulder CO)

[NTI85] "The OSI Network Layer Addressing Scheme, Its Implications, and Considerations for Implementation", NTIA Report 85-186, (available from NTIA, Boulder CO)

[RFC85] Mills, David, "Internet Delay Experiments," RFC889,

December 1983 (available from the Network Information Center).

[SPI82] Spirn, Jeffery R., "Network Modeling with Bursty Traffic and Finite Buffer Space," Performance Evaluation Review, vol. 2, no. 1, April 1982.

[SPI84] Spirn, Jeffery R., Jade Chien, William Hawe, "Bursty Traffic Local Area Network Modeling," IEEE Journal on Selected Areas in Communications, vol. SAC-2, no. 1, January 1984.